

OpenOffice.org 1.1

Developer's Guide

June 2003

This documentation is distributed under licenses restricting its use. You may make copies of and redistribute it, but you may not modify or make derivative works of this documentation without prior written authorization of Sun and its licensors, if any.

Copyright 2003 Sun Microsystems, Inc.

Contents

1 Reader's Guide 25

1.1 What This Manual Covers 25
1.2 How This Book is Organized 25
1.3 OpenOffice.org Version History 26
1.4 Related documentation 26
1.5 Conventions 27
1.6 Acknowledgments 27

2 First Steps 29

2.1 Programming with UNO 29
2.2 Fields of Application for UNO 29
2.3 Getting Started 30
2.3.1 Required Files 30
2.3.2 Installation Sets 30
2.3.3 Configuration 31
Enable Java in OpenOffice.org 31
Use Java UNO class files 31
Make the office listen 31
Add the API Reference to your IDE 32
2.3.4 First Connection 33
Getting Connected 33
Service Managers 34
Failed Connections 36
2.4 How to get Objects in OpenOffice.org 37
2.5 Working with Objects 38
2.5.1 Services 38
Using Interfaces 41
Using Properties 42
2.5.2 Example: Working with a Spreadsheet Document 43
2.5.3 Common Types 44
Simple Types 44
Strings 45
Enum Types and Groups of Constants 45
2.5.4 Struct 46
2.5.5 Any 46
2.5.6 Sequence 48
2.5.7 Element Access 49
Name Access 51
Index Access 52
Enumeration Access 52

2.6 How do I know Which Type I Have? 53
2.7 Example: Hello Text, Hello Table, Hello Shape 54
2.7.1 Common Mechanisms for Text, Tables and Drawings 54
2.7.2 Creating Text, Tables and Drawing Shapes 59
Text, Tables and Drawings in Writer 59
Text, Tables and Drawings in Calc 60
Drawings and Text in Draw 61

3 Professional UNO 63

3.1 Introduction 63
3.2 API Concepts 64
3.2.1 Data Types 64
Simple Types 64
The Any Type 65
Interfaces 65
Services 67
Structs 71
Predefined Values 71
Sequences 72
Modules 72
Exceptions 73
Singletons 73
3.2.2 Understanding the API Reference 74
Specification, Implementation and Instances 74
Object Composition 75
3.3 UNO Concepts 75
3.3.1 UNO Interprocess Connections 75
Starting OpenOffice.org in Listening Mode 75
Importing a UNO Object 76
Characteristics of the Interprocess Bridge 77
Opening a Connection 78
Creating the Bridge 80
Closing a Connection 81
Example: A Connection Aware Client 82
3.3.2 Service Manager and Component Context 84
Service Manager 84
Component Context 85
3.3.3 Using UNO Interfaces 89
3.3.4 Properties 91
3.3.5 Collections and Containers 95
3.3.6 Event Model 97
3.3.7 Exception Handling 98
User-Defined Exceptions 98

Runtime Exceptions ... 99
Good Exception Handling ... 99
3.3.8 Lifetime of UNO Objects ... 100
acquire() and release() ... 101
The XComponent Interface ... 101
Children of the XEventListener Interface ... 103
Weak Objects and References ... 104
Differences Between the Lifetime of C++ and Java Objects ... 104
3.3.9 Object Identity ... 106
3.4 UNO Language Bindings ... 106
3.4.1 Java Language Binding ... 107
Getting a Service Manager ... 107
Handling Interfaces ... 108
Type Mappings ... 109
3.4.2 UNO C++ Binding ... 117
Library Overview ... 118
System Abstraction Layer ... 119
File Access ... 119
Threadsafe Reference Counting ... 119
Threads and Thread Synchronization ... 120
Establishing Interprocess Connections ... 121
Type Mappings ... 122
Using Weak References ... 126
Exception Handling in C++ ... 127
3.4.3 OpenOffice.org Basic ... 128
Handling UNO Objects ... 129
Mapping of UNO and Basic Types ... 135
Case Sensitivity ... 141
Exception Handling ... 142
Listeners ... 143
3.4.4 Automation Bridge ... 145
Introduction ... 145
Requirements ... 146
A Quick Tour ... 146
The Service Manager Component ... 148
Using UNO from Automation ... 150
Type Mappings ... 156
Automation Objects with UNO Interfaces ... 167
DCOM ... 170
The Bridge Services ... 172
Unsupported COM Features ... 175

4 Writing UNO Components 177

4.1 Required Files ... 178
4.2 Using UNOIDL to Specify New Components ... 179
4.2.1 Writing the Specification ... 179
Preprocessing ... 180
Grouping Definitions in Modules ... 181
Fundamental Types ... 181
Defining an Interface ... 182
Defining a Service ... 185
Defining a Sequence ... 187
Defining a Struct ... 187
Defining an Exception ... 188
Predefining Values ... 189
Using Comments ... 190
Singleton ... 191
Reserved Types ... 191
4.2.2 Generating Source Code from UNOIDL Definitions ... 192
4.3 Component Architecture ... 193
4.4 Core Interfaces to Implement ... 194
4.4.1 XInterface ... 196
Requirements for queryInterface() ... 197
Reference Counting ... 197
4.4.2 XTypeProvider ... 197
Provided Types ... 198
ImplementationID ... 198
4.4.3 XServiceInfo ... 198
Implementation Name ... 198
Supported Service Names ... 199
4.4.4 XWeak ... 199
4.4.5 XComponent ... 200
Disposing of an XComponent ... 200
4.4.6 XInitialization ... 200
4.4.7 XMain ... 201
4.4.8 XAggregation ... 201
4.4.9 XUnoTunnel ... 201
4.5 Simple Component in Java ... 202
4.5.1 Class Definition with Helper Classes ... 203
XInterface, XTypeProvider and XWeak ... 203
XServiceInfo ... 203
4.5.2 Implementing your own Interfaces ... 204
4.5.3 Providing a Single Factory Using Helper Method ... 205
4.5.4 Write Registration Info Using Helper Method ... 206
4.5.5 Implementing without Helpers ... 207
XInterface ... 207
XTypeProvider ... 208

XComponent 208
4.5.6 Storing the Service Manager for Further Use 209
4.5.7 Create Instance with Arguments 209
4.5.8 Possible Structures for Java Components 210
One Implementation per Component File 210
Multiple Implementations per Component File 212
4.5.9 Running and Debugging Java Components 214
Registration 214
Debugging 215
The Java Environment in OpenOffice.org 216
Troubleshooting 217
4.6 C++ Component 219
4.6.1 Class Definition with Helper Template Classes 220
XInterface, XTypeProvider and XWeak 220
XServiceInfo 221
4.6.2 Implementing your own Interfaces 221
4.6.3 Providing a Single Factory Using a Helper Method 222
4.6.4 Write Registration Info Using Helper Method 223
4.6.5 Provide Implementation Environment 224
4.6.6 Implementing without Helpers 224
XInterface Implementation 224
XTypeProvider Implementation 225
Providing a Single Factory 226
Write Registration Info 227
4.6.7 Storing the Service Manager for Further Use 227
4.6.8 Create Instance with Arguments 228
4.6.9 Multiple Components in One Dynamic Link Library 228
4.6.10 Building and Testing C++ Components 228
Build Process 228
Test Registration and Use 229
4.7 Integrating Components into OpenOffice.org 230
4.7.1 Protocol Handler 232
Overview 232
Implementation 233
Configuration 241
Installation 243
4.7.2 Jobs 243
Overview 243
Execution Environment 245
Implementation 246
Initialization 248
Returning Results 250
Configuration 251
Installation 253

Using the vnd.sun.star.jobs: URL Schema ... 253
List of supported Events ... 255
4.7.3 Add-Ons ... 256
Overview ... 257
Guidelines ... 258
Configuration ... 258
Installation ... 269
4.7.4 Disable Commands ... 270
Configuration ... 272
Disabling Commands at Runtime ... 273
4.8 File Naming Conventions ... 276
4.9 Deployment Options for Components ... 278
4.9.1 UNO Package Installation ... 278
Package Structure ... 279
Path Settings ... 280
Additional Options ... 281
4.9.2 Background: UNO Registries ... 281
UNO Type Library ... 282
Component Registration ... 282
4.9.3 Command Line Registry Tools ... 283
Component Registration Tool ... 283
UNO Type Library Tools ... 284
4.9.4 Manual Component Installation ... 285
Manually Merging a Registry and Adding it to uno.ini or soffice.ini ... 285
Alternatives ... 286
4.9.5 Bootstrapping a Service Manager ... 286
4.9.6 Special Service Manager Configurations ... 288
Dynamically Modifying the Service Manager ... 288
Creating a ServiceManager from a Given Registry File ... 290
4.10 The UNO Executable ... 290
Standalone Use Case ... 291
Server Use Case ... 293
Using the uno Executable ... 294

5 Advanced UNO 295

5.1 Choosing an Implementation Language ... 295
5.1.1 Supported Programming Environments ... 295
Java ... 296
C++ ... 296
OpenOffice.org Basic ... 296
OLE Automation Bridge ... 297
Python ... 297
5.1.2 Use Cases ... 297

Java ... 297
C++ ... 297
OpenOffice.org Basic ... 298
OLE Automation ... 298
Python ... 298
5.1.3 Recommendation ... 298
5.2 Language Bindings ... 298
5.2.1 Implementing UNO Language Bindings ... 299
Overview of Language Bindings and Bridges ... 299
Implementation Options ... 300
5.2.2 UNO C++ bridges ... 301
Binary UNO Interfaces ... 302
C++ Proxy ... 303
Binary UNO Proxy ... 304
Additional Hints ... 305
5.2.3 UNO Reflection API ... 306
XTypeProvider Interface ... 306
Converter Service ... 306
CoreReflection Service ... 306
5.2.4 XInvocation Bridge ... 310
Scripting Existing UNO Objects ... 310
Implementing UNO objects ... 313
Example: Python Bridge PyUNO ... 314
5.2.5 Implementation Loader ... 316
Shared Library Loader ... 318
Bridges ... 318
5.2.6 Help with New Language Bindings ... 319
5.3 Differences Between UNO and Corba ... 319
5.4 UNO Design Patterns and Coding Styles ... 321
5.4.1 Double-Checked Locking ... 321

6 Office Development 325

6.1 OpenOffice.org Application Environment ... 325
6.1.1 Overview ... 325
Desktop Environment ... 326
Framework API ... 327
6.1.2 Using the Desktop ... 333
6.1.3 Using the Component Framework ... 337
Getting Frames, Controllers and Models from Each Other ... 338
Frames ... 339
Controllers ... 344
Models ... 346
Window Interfaces ... 349

6.1.4 Creating Frames Manually ... 350
6.1.5 Handling Documents ... 352
Loading Documents ... 352
Closing Documents ... 360
Storing Documents ... 365
Printing Documents ... 367
6.1.6 Using the Dispatch Framework ... 367
Command URL ... 367
Processing Chain ... 368
Dispatch Process ... 369
Dispatch Results ... 373
Dispatch Interception ... 373
6.1.7 Java Window Integration ... 375
The Window Handle ... 375
Using the Window Handle ... 376
More Remote Problems ... 378
6.2 Common Application Features ... 378
6.2.1 Clipboard ... 378
Using the Clipboard ... 379
OpenOffice.org Clipboard Data Formats ... 383
6.2.2 Internationalization ... 383
Introduction ... 383
Overview and Using the API ... 384
Implementing a New Locale ... 386
6.2.3 Linguistics ... 394
Services Overview ... 394
Using Spellchecker ... 396
Using Hyphenator ... 397
Using Thesaurus ... 399
Events ... 400
Implementing a Spell Checker ... 401
Implementing a Hyphenator ... 403
Implementing a Thesaurus ... 403
6.2.4 Integrating Import and Export Filters ... 404
Approaches ... 404
Document API Filter Development ... 405
XML Based Filter Development ... 419
6.2.5 Number Formats ... 427
Managing Number Formats ... 427
Applying Number Formats ... 428
6.2.6 Document Events ... 430
6.2.7 Path Organization ... 435
Path Settings ... 435
Path Variables ... 442

6.2.8 OpenOffice.org Single Sign-On API ... 451
Overview ... 451
Implementing the OpenOffice.org SSO API ... 452

7 Text Documents 457

7.1 Overview ... 457
7.1.1 Example: Fields in a Template ... 460
7.1.2 Example: Visible Cursor Position ... 461
7.2 Handling Text Document Files ... 463
7.2.1 Creating and Loading Text Documents ... 463
7.2.2 Saving Text Documents ... 464
Storing ... 464
Exporting ... 464
7.2.3 Printing Text Documents ... 465
Printer and Print Job Settings ... 465
Printing Multiple Pages on one Page ... 466
7.3 Working with Text Documents ... 467
7.3.1 Word Processing ... 467
Editing Text ... 467
Iterating over Text ... 471
Inserting a Paragraph where no Cursor can go ... 473
Sorting Text ... 473
Inserting Text Files ... 473
Auto Text ... 473
7.3.2 Formatting ... 474
7.3.3 Navigating ... 481
Cursors ... 481
Locating Text Contents ... 481
Search and Replace ... 482
7.3.4 Tables ... 485
Table Architecture ... 485
Named Table Cells in Rows, Columns and the Table Cursor ... 488
Indexed Cells and Cell Ranges ... 490
Table Naming, Sorting, Charting and Autoformatting ... 491
Text Table Properties ... 491
Inserting Tables ... 492
Accessing Existing Tables ... 497
7.3.5 Text Fields ... 497
7.3.6 Bookmarks ... 503
7.3.7 Indexes and Index Marks ... 504
Indexes ... 504
Index marks ... 507
7.3.8 Reference Marks ... 508

7.3.9 Footnotes and Endnotes ... 509
7.3.10 Shape Objects in Text ... 511
Base Frames vs. Drawing Shapes ... 511
Text Frames ... 514
Embedded Objects ... 516
Graphic Objects ... 516
Drawing Shapes ... 517
7.3.11 Redline ... 519
7.3.12 Ruby ... 520
7.4 Overall Document Features ... 520
7.4.1 Styles ... 520
Character Styles ... 522
Paragraph Styles ... 523
Frame Styles ... 523
Page Styles ... 523
Numbering Styles ... 523
7.4.2 Settings ... 525
General Document Information ... 525
Document Properties ... 525
Creating Default Settings ... 526
Creating Document Settings ... 526
7.4.3 Line Numbering and Outline Numbering ... 526
Paragraph and Outline Numbering ... 526
Line Numbering ... 529
Number Formats ... 529
7.4.4 Text Sections ... 529
7.4.5 Page Layout ... 530
7.4.6 Columns ... 531
7.4.7 Link targets ... 533
7.5 Text Document Controller ... 533
7.5.1 TextView ... 534
7.5.2 TextViewCursor ... 535

8 Spreadsheet Documents 537

8.1 Overview ... 537
8.1.1 Example: Adding a New Spreadsheet ... 539
8.1.2 Example: Editing Spreadsheet Cells ... 540
8.2 Handling Spreadsheet Document Files ... 540
8.2.1 Creating and Loading Spreadsheet Documents ... 540
8.2.2 Saving Spreadsheet Documents ... 541
Storing ... 541
Exporting ... 542
Filter Options ... 542

8.2.3 Printing Spreadsheet Documents ... 545
Printer and Print Job Settings ... 545
Page Breaks and Scaling for Printout ... 546
Print Areas ... 546
8.3 Working with Spreadsheet Documents ... 547
8.3.1 Document Structure ... 547
Spreadsheet Document ... 547
Spreadsheet Services - Overview ... 551
Spreadsheet ... 562
Cell Ranges ... 564
Cells ... 571
Cell Ranges and Cells Container ... 575
Columns and Rows ... 578
8.3.2 Formatting ... 580
Cell Formatting ... 580
Character and Paragraph Format ... 580
Indentation ... 581
Equally Formatted Cell Ranges ... 581
Table Auto Formats ... 585
Conditional Formats ... 589
8.3.3 Navigating ... 590
Cell Cursor ... 592
Referencing Ranges by Name ... 595
Named Ranges ... 596
Label Ranges ... 598
Querying for Cells with Specific Properties ... 599
Search and Replace ... 602
8.3.4 Sorting ... 602
Table Sort Descriptor ... 602
8.3.5 Database Operations ... 604
Filtering ... 605
Subtotals ... 607
Database Import ... 608
Database Ranges ... 610
8.3.6 Linking External Data ... 610
Sheet Links ... 611
Cell Area Links ... 611
DDE Links ... 613
8.3.7 DataPilot ... 613
DataPilot Tables ... 613
DataPilot Sources ... 617
8.3.8 Protecting Spreadsheets ... 626
8.3.9 Sheet Outline ... 627
8.3.10 Detective ... 627

8.3.11 Other Table Operations ... 627
Data Validation ... 627
Data Consolidation ... 629
Charts ... 630
Scenarios ... 631
8.4 Overall Document Features ... 634
8.4.1 Styles ... 634
Cell Styles ... 635
Page Styles ... 636
8.4.2 Function Handling ... 637
Calculating Function Results ... 637
Information about Functions ... 639
Recently Used Functions ... 641
8.4.3 Settings ... 641
8.5 Spreadsheet Document Controller ... 642
8.5.1 Spreadsheet View ... 642
8.5.2 View Panes ... 644
8.5.3 View Settings ... 645
8.5.4 Range Selection ... 645
8.6 Spreadsheet Add-Ins ... 648
8.6.1 Function Descriptions ... 648
8.6.2 Service Names ... 649
8.6.3 Compatibility Names ... 649
8.6.4 Custom Functions ... 649
8.6.5 Variable Results ... 650

9 Drawing Documents and Presentation Documents 653

9.1 Overview ... 653
9.1.1 Example: Creating a Simple Organizational Chart ... 655
9.2 Handling Drawing Document Files ... 657
9.2.1 Creating and Loading Drawing Documents ... 657
9.2.2 Saving Drawing Documents ... 658
Storing ... 658
Exporting ... 659
Filter Options ... 660
9.2.3 Printing Drawing Documents ... 661
Printer and Print Job Settings ... 661
Special Print Settings ... 663
9.3 Working with Drawing Documents ... 663
9.3.1 Drawing Document ... 663
Document Structure ... 663
Page Handling ... 664
Page Partitioning ... 665

9.3.2 Shapes ... 665
Bezier Shapes ... 671
Shape Operations ... 674
9.3.3 Inserting Files ... 686
9.3.4 Navigating ... 686
9.4 Handling Presentation Document Files ... 686
9.4.1 Creating and Loading Presentation Documents ... 686
9.4.2 Printing Presentation Documents ... 687
9.5 Working with Presentation Documents ... 687
9.5.1 Presentation Document ... 687
9.5.2 Presentation Settings ... 689
Custom Slide Show ... 690
Presentation Effects ... 692
Slide Transition ... 692
Animations and Interactions ... 693
9.6 Overall Document Features ... 697
9.6.1 Styles ... 697
Graphics Styles ... 697
Presentation Styles ... 699
9.6.2 Settings ... 700
9.6.3 Page Formatting ... 701
9.7 Drawing and Presentation Document Controller ... 702
9.7.1 Setting the Current Page, Using the Selection ... 702
9.7.2 Zooming ... 703
9.7.3 Other Drawing-Specific View Settings ... 703

10 Charts 705

10.1 Overview ... 705
10.2 Handling Chart Documents ... 705
10.2.1 Creating Charts ... 705
Creating and Adding a Chart to a Spreadsheet ... 705
Creating a Chart OLE Object in Draw and Impress ... 706
Setting the Chart Type ... 708
10.2.2 Accessing Existing Chart Documents ... 708
10.3 Working with Charts ... 709
10.3.1 Document Structure ... 709
10.3.2 Data Access ... 710
10.3.3 Chart Document Parts ... 713
Common Parts of all Chart Types ... 713
Features of Special Chart Types ... 717
10.4 Chart Document Controller ... 720
10.5 Chart Add-Ins ... 720
10.5.1 Prerequisites ... 720

10.5.2 How Add-Ins work ... 720
10.5.3 How to Apply an Add-In to a Chart Document ... 722

11 OpenOffice.org Basic and Dialogs 725

11.1 First Steps with OpenOffice.org Basic ... 726
Step By Step Tutorial ... 726
A Simple Dialog ... 730
11.2 OpenOffice.org Basic IDE ... 735
11.2.1 Managing Basic and Dialog Libraries ... 736
Macro Dialog ... 736
Macro Organizer Dialog ... 738
11.2.2 Basic IDE Window ... 743
Basic Source Editor and Debugger ... 745
Dialog Editor ... 748
11.2.3 Assigning Macros to GUI Events ... 754
11.3 Features of OpenOffice.org Basic ... 757
11.3.1 Functional Range Overview ... 757
Screen I/O Functions ... 757
File I/O ... 757
Date and Time Functions ... 758
Numeric Functions ... 759
String Functions ... 759
Specific UNO Functions ... 760
11.3.2 Accessing the UNO API ... 760
StarDesktop ... 760
ThisComponent ... 760
11.3.3 Special Behavior of OpenOffice.org Basic ... 762
Threads ... 762
Rescheduling ... 762
11.4 Advanced Library Organization ... 764
11.4.1 General Structure ... 764
11.4.2 Accessing Libraries from Basic ... 765
Library Container Properties in Basic ... 765
Loading Libraries ... 766
Library Container API ... 767
11.4.3 Variable Scopes ... 769
11.5 Programming Dialogs and Dialog Controls ... 770
11.5.1 Dialog Handling ... 770
Showing a Dialog ... 770
Getting the Dialog Model ... 771
Dialog as Control Container ... 771
Dialog Properties ... 772
Common Properties ... 772

Multi-Page Dialogs ... 772
11.5.2 Dialog Controls ... 773
Command Button ... 773
Image Control ... 773
Check Box ... 774
Option Button ... 774
Label Field ... 774
Text Field ... 775
List Box ... 776
Combo Box ... 776
Horizontal/Vertical Scroll Bar ... 777
Group Box ... 778
Progress Bar ... 778
Horizontal/Vertical Line ... 779
Date Field ... 779
Time Field ... 779
Numeric Field ... 780
Currency Field ... 780
Formatted Field ... 780
Pattern Field ... 780
File Control ... 781
11.6 Creating Dialogs at Runtime ... 781
11.7 Library File Structure ... 784
11.7.1 Application Library Container ... 785
11.7.2 Document Library Container ... 787
11.8 Library Deployment ... 789
Package Structure ... 789
Path Settings ... 790
Additional Options ... 791

12 Database Access 793

12.1 Overview ... 793
12.1.1 Capabilities ... 793
Platform Independence ... 793
Functioning of the OpenOffice.org API Database Integration ... 793
Integration with OpenOffice.org API ... 794
12.1.2 Architecture ... 794
12.1.3 Example: Querying the Bibliography Database ... 794
12.2 Data Sources in OpenOffice.org API ... 796
12.2.1 DatabaseContext ... 796
12.2.2 DataSources ... 798
The DataSource Service ... 798
Queries ... 800

Forms and Other Links ... 806
Tables and Columns ... 807
12.2.3 Connections ... 811
Understanding Connections ... 811
Connecting Using the DriverManager and a Database URL ... 814
Connecting Through a Specific Driver ... 816
Driver Specifics ... 816
Connection Pooling ... 820
Piggyback Connections ... 822
12.3 Manipulating Data ... 822
12.3.1 The RowSet Service ... 822
Usage ... 822
Events and Other Notifications ... 826
Clones of the RowSet Service ... 828
12.3.2 Statements ... 829
Creating Statements ... 829
Inserting and Updating Data ... 830
Getting Data from a Table ... 831
12.3.3 Result Sets ... 832
Retrieving Values from Result Sets ... 835
Moving the Result Set Cursor ... 835
Using the getXXX Methods ... 836
Scrollable Result Sets ... 838
Modifiable Result Sets ... 840
Update ... 840
Insert ... 842
Delete ... 843
Seeing Changes in Result Sets ... 844
12.3.4 ResultSetMetaData ... 845
12.3.5 Using Prepared Statements ... 845
When to Use a PreparedStatement Object ... 845
Creating a PreparedStatement Object ... 846
Supplying Values for PreparedStatement Parameters ... 846
12.3.6 PreparedStatement From DataSource Queries ... 847
12.4 Database Design ... 848
12.4.1 Retrieving Information about a Database ... 848
Retrieving General Information ... 848
Determining Feature Support ... 849
Database Limits ... 849
SQL Objects and their Attributes ... 849
12.4.2 Using DDL to Change the Database Design ... 850
12.4.3 Using SDBCX to Access the Database Design ... 853
The Extension Layer SDBCX ... 853
Catalog Service ... 854

Table Service ... 855
Column Service ... 858
Index Service ... 859
Key Service ... 861
View Service ... 863
Group Service ... 863
User Service ... 865
The Descriptor Pattern ... 865
Adding an Index ... 867
Creating a User ... 867
Adding a Group ... 868
12.5 Using DBMS Features ... 868
12.5.1 Transaction Handling ... 868
12.5.2 Stored Procedures ... 869
12.6 Writing Database Drivers ... 870
12.6.1 SDBC Driver ... 870
12.6.2 Driver Service ... 871
12.6.3 Connection Service ... 872
12.6.4 XDatabaseMetaData Interface ... 873
12.6.5 Statements ... 874
PreparedStatement ... 875
Result Set ... 875
12.6.6 Support Scalar Functions ... 875
Open Group CLI Numeric Functions ... 875
Open Group CLI String Functions ... 876
Open Group CLI Time and Date Functions ... 877
Open Group CLI System Functions ... 877
Open Group CLI Conversion Functions ... 878
Handling Unsupported Functionality ... 878

13 Forms 879

13.1 Introduction ... 879
13.2 Models and Views ... 879
13.2.1 The Model-View Paradigm ... 879
13.2.2 Models and Views for Form Controls ... 880
13.2.3 Model-View Interaction ... 881
13.2.4 Form Layer Views ... 881
View Modes ... 881
Locating Controls ... 882
Focussing Controls ... 882
13.3 Form Elements in the Document Model ... 882
13.3.1 A Hierarchy of Models ... 883
FormComponent Service ... 883

FormComponents Service 883
Logical Forms 884
Forms Container 884
Form Control Models 885
13.3.2 Control Models and Shapes 885
Programmatic Creation of Controls 886
13.4 Form Components 888
13.4.1 Basics 888
Control Models 888
Forms 890
13.4.2 HTML Forms 890
13.5 Data Awareness 890
13.5.1 Forms 891
Forms as Row Sets 891
Loadable Forms 891
Sub Forms 891
Filtering and Sorting 893
Parameters 893
13.5.2 Data Aware Controls 894
Control Models as Bound Components 895
Committing Controls 896
13.6 Common Tasks 897
13.6.1 Initializing Bound Controls 897
13.6.2 Automatic Key Generation 898
13.6.3 Data Validation 899

14 Universal Content Broker 901

14.1 Overview 901
14.1.1 Capabilities 901
14.1.2 Architecture 901
14.2 Services and Interfaces 902
14.3 Content Providers 904
14.4 Using the UCB API 904
14.4.1 Instantiating the UCB 905
14.4.2 Accessing a UCB Content 905
14.4.3 Executing Content Commands 906
14.4.4 Obtaining Content Properties 907
14.4.5 Setting Content Properties 908
14.4.6 Folders 909
Accessing the Children of a Folder 909
14.4.7 Documents 911
Reading a Document Content 911
Storing a Document Content 913

14.4.8 Managing Contents ... 913
Creating ... 913
Deleting ... 915
Copying, Moving and Linking ... 916
14.4.9 UCP Registration Information ... 917
14.4.10 Unconfigured UCBs ... 917
14.4.11 Preconfigured UCBs ... 919
14.4.12 Content Provider Proxies ... 920

15 Configuration Management 923

15.1 Overview ... 923
15.1.1 Capabilities ... 923
15.1.2 Architecture ... 923
15.2 Object Model ... 926
15.3 Configuration Data Sources ... 928
15.3.1 Connecting to a Data Source ... 928
15.3.2 Using a Data Source ... 930
15.4 Accessing Configuration Data ... 932
15.4.1 Reading Configuration Data ... 932
15.4.2 Updating Configuration Data ... 935
15.5 Customizing Configuration Data ... 943
15.6 Adding a Backend Data Store ... 944

16 Office Bean 945

16.1 Introduction ... 945
16.2 Overview of the OfficeBean API ... 946
16.2.1 OfficeConnection Interface ... 947
16.2.2 OfficeWindow Interface ... 948
16.2.3 ContainerFactory Interface ... 948
16.3 LocalOfficeConnection and LocalOfficeWindow ... 948
16.4 Configuring the OfficeBean ... 949
16.4.1 Default Configuration ... 949
16.4.2 Customized Configuration ... 950
16.5 Using the OfficeBean ... 951
16.5.1 SimpleBean Example ... 953
Using SimpleBean ... 953
SimpleBean Internals ... 955
16.5.2 OfficeWriterBean Example ... 957

17 Accessibility 959

17.1 Overview ... 959
17.2 Bridges ... 960
17.3 Accessibility Tree ... 960

17.4 Content Information ... 961
17.5 Listeners and Broadcasters ... 961
17.6 Implementing Accessible Objects ... 962
17.6.1 Implementation Rules ... 962
17.6.2 Services ... 962
17.7 Using the Accessibility API ... 963
17.7.1 A Simple Screen Reader ... 963
Features ... 964
Class Overview ... 965
Putting the Accessibility Interfaces to Work ... 966

Appendix A: OpenOffice.org API-Design-Guidelines 979

A.1 General Design Rules ... 979
A.1.1 Universality ... 979
A.1.2 Orthogonality ... 980
A.1.3 Inheritance ... 980
A.1.4 Uniformity ... 980
A.1.5 Correct English ... 980
A.2 Definition of API Elements ... 980
A.2.1 Attributes ... 980
A.2.2 Methods ... 981
A.2.3 Interfaces ... 983
A.2.4 Properties ... 983
A.2.5 Events ... 984
A.2.6 Services ... 984
A.2.7 Exceptions ... 985
A.2.8 Enums ... 985
A.2.9 Typedefs ... 986
A.2.10 Structs ... 986
A.2.11 Parameter ... 986
A.3 Special Cases ... 987
A.4 Abbreviations ... 987
A.5 Source Files and Types ... 988

Appendix B: IDL Documentation Guidelines 989

B.1 Introduction ... 989
B.1.1 Process ... 989
B.1.2 File Assembly ... 989
B.1.3 Readable & Editable Structure ... 990
B.1.4 Contents ... 990
B.2 File structure ... 990
B.2.1 General ... 990
B.2.2 File-Header ... 991

B.2.3 File-Footer ... 992
B.3 Element Documentation ... 992
B.3.1 General Element Documentation ... 992
B.3.2 Example for a Major Element Documentation ... 993
B.3.3 Example for a Minor Element Documentation ... 994
B.4 Markups and Tags ... 994
B.4.1 Special Markups ... 994
B.4.2 Special Documentation Tags ... 996
B.4.3 Useful XHTML Tags ... 997

Appendix C: Universal Content Providers 1001

C.1 The Hierarchy Content Provider ... 1001
C.1.1 Preface ... 1001
C.1.2 HCP Contents ... 1001
C.1.3 Creation of New HCP Content ... 1002
C.1.4 URL Scheme for HCP Contents ... 1002
C.1.5 Commands and Properties ... 1003
C.2 The File Content Provider ... 1003
C.2.1 Preface ... 1003
C.2.2 File Contents ... 1003
C.2.3 Creation of New File Contents ... 1004
C.2.4 URL Schemes for File Contents ... 1004
C.2.5 Commands and Properties ... 1005
C.3 The FTP Content Provider ... 1005
C.3.1 Preface ... 1005
C.3.2 FTP Contents ... 1005
C.3.3 Creation of New FTP Content ... 1006
C.3.4 URL Scheme for FTP Contents ... 1006
C.3.5 Commands and Properties ... 1007
C.4 The WebDAV Content Provider ... 1008
C.4.1 Preface ... 1008
C.4.2 DCP Contents ... 1008
C.4.3 Creation of New DCP Contents ... 1009
C.4.4 Authentication ... 1009
C.4.5 Property Handling ... 1009
C.4.6 URL Scheme for DCP Contents ... 1010
C.4.7 Commands and Properties ... 1011
C.5 The Package Content Provider ... 1011
C.5.1 Preface ... 1011
C.5.2 PCP Contents ... 1011
C.5.3 Creation of New PCP Contents ... 1012
C.5.4 URL Scheme for PCP Contents ... 1012
C.5.5 Commands and Properties ... 1013

C.6 The Help Content Provider 1013
C.6.1 Preface 1013
C.6.2 Help Content Provider Contents 1014
C.6.3 URL Scheme for Help Contents 1014
C.6.4 Properties and Commands 1015

Appendix D: UNOIDL Syntax Specification **1019**

Glossary **1023**

Index **1041**

1 Reader's Guide

1.1 What This Manual Covers

This manual describes how to write programs using the component technology UNO (Universal Network Objects) with OpenOffice.org.

Most examples provided are written in Java. As well as Java, the language binding for C++, the UNO access for OpenOffice.org Basic and the OLE Automation bridge that uses OpenOffice.org through Microsoft's component technology COM/DCOM is described.

1.2 How This Book is Organized

First Steps
The First Steps chapter describes the setting up of a Java UNO development environment to achieve the solutions you need. At the end of this chapter, you will be equipped with the essentials required for the following chapters about the OpenOffice.org applications.

Professional UNO Projects
This chapter introduces API and UNO concepts and explains the specifics of the programming languages and technologies that can be used with UNO. It will help you to write industrial-strength UNO programs, use one of the languages besides Java or improve your understanding of the API reference.

Writing UNO Components
This chapter describes how to write UNO components. It also provides an insight into the UNOIDL (UNO Interface Definition Language) language and the inner workings of service manager. Before beginning this chapter, you should be familiar with the First Steps and Professional UNO chapters.

Advanced UNO
This chapter describes the technical basis of UNO, how the language bindings and bridges work, how the service manager goes about its tasks and what the core reflection actually does.

Office Development
This chapter describes the application framework of the OpenOffice.org application that includes how the OpenOffice.org API deals with the OpenOffice.org application and the features available across all parts of OpenOffice.org.

Text Documents - Spreadsheet Documents - Drawings and Presentations – Chart
These chapters describes how OpenOffice.org revolves around documents. These chapters teach you what to do with these documents programmatically.

Basic and Dialogs
This chapter provides the functionality to create and manage Basic macros and dialogs.

Database Access
This chapter describes how you can take advantage of this capability in your own projects OpenOffice.org can connect to databases in a universal manner.

Forms
This chapter describes how OpenOffice.org documents contain form controls that are programmed using an event-driven programming model. The Forms chapter shows you how to enhance your documents with controls for data input.

UCB
This chapter describes how the Universal Content Broker is the generic resource access service used by the entire office application. It handles not only files and directories, but hierarchic and non-hierarchic contents, in general.

OpenOffice.org Configuration
This chapter decribes how the OpenOffice.org API offers access to the office configuration options that is found in the Tools – Options dialog.

OfficeBean
This chapter describes how the OfficeBean Java Bean component allows the developer to integrate office functionality in Java applications.

1.3 OpenOffice.org Version History

OpenOffice.org exists in two versions *www.openoffice.org*

OpenOffice.org - an open source edition

StarOffice and StarSuite - "branded" editions derived from OpenOffice.org

In 2000, Sun Microsystems released the source code of their current developer version of StarOffice on *www.openoffice.org*, and made the ongoing development process public. Sun's development team, which developed StarOffice, continued its work on *www.openoffice.org*, and developers from all over the world joined them to port, translate, repair bugs and discuss future plans. StarOffice 6.0 and OpenOffice.org 1.0, which were released in spring 2002, share the same code basis.

1.4 Related documentation

The api and udk projects on *www.openoffice.org* have related documentation, examples and FAQs (frequently asked questions) on the OpenOffice.org API. Most important are probably the references, you can find them at *api.openoffice.org* or udk.*openoffice.org*.

- The API Reference covers the programmable features of OpenOffice.org.
- The Java Reference describes the features of the Java UNO runtime environment.
- The C++ Reference is about the C++ language binding.

1.5 Conventions

This book uses the following formatting conventions:

- **Bold** refers to the keys on the keyboard or elements of a user interface, such as the **OK** button or **File** menu.
- *Italics* are used for emphasis and to signify the first use of a term. Italics are also used for web sites, file and directory names and email addresses.
- `Courier New` is used in all Code Listings and for everything that is typed when programming.

1.6 Acknowledgments

A publication like this can never be the work of a single person – it is the result of tremendous team effort. Of course, the OpenOffice.org/StarOffice development team played the most important role by creating the API in the first place. The knowledge and experience of this team will be documented here. Furthermore, there were several devoted individuals who contributed to making this documentation reality.

First of all, we would like to thank Ralf Kuhnert and Dietrich Schulten. Using their technical expertise and articulate mode of expression, they accomplished the challenging task of gathering the weatlth of API knowledge from the minds of the developers and transforming it into an understandable document.

Many reviewers were involved in the creation of this documentation. Special thanks go to Michael Hönnig who was one of the few who reviewed almost every single word. His input also played a decisive role in how the documentation was structured. A big thank you also goes to Diane O'Brien for taking on the daunting task of reviewing the final draft and providing us with extensive feedback at such short notice.

When looking at the diagrams and graphics, it is clear that a creative person with the right touch for design and aesthetics was involved. Many thanks, therefore, are due Stella Schulze who redrew all of the diagrams and graphics from the originals supplied by various developers. We also thank Svante Schubert who converted the original XML file format into HTML pages and was most patient with us in spite of our demands and changes. Special thanks also to Jörg Heilig, who made this whole project possible.

Jürgen would like to thank Götz Wohlberg for all his help in getting the right people involved and making sure things ran smoothly.

Götz would like to thank Jürgen Schmidt for his never-ending energy to hold everything together and for pushing the contributors in the right direction. He can be considered as the heart of the opus because of his guidance and endurance throughout the entire project.

We would like to take this opportunity to thank all these people – and anyone else we forgot! – for their support.

Jürgen Schmidt, Götz Wohlberg

2 First Steps

This chapter shows you the first steps when using the OpenOffice.org API. Following these steps is essential to understand and use the chapters about OpenOffice.org documents such as 7 Text Documents, 8 Spreadsheet Documents and 9 Drawing. After you have successfully done the first steps, you can go directly to the other chapters of this manual.

The focus of the first steps will be Java, but other languages are covered as well. If you want to use OpenOffice.org Basic afterwards, please refer to the chapters 11.1 Basic and Dialogs - First Steps with OpenOffice.org Basic and 3.4.3 Professional UNO - UNO Language Bindings - OpenOffice.org Basic. The usage of C++ is described in 3.4.2 Professional UNO - UNO Language Bindings - UNO C++ Binding.

2.1 Programming with UNO

UNO (pronounced ['ju:nou]) stands for Universal Network Objects and is the base component technology for OpenOffice.org. You can utilize and write components that interact across languages, component technologies, computer platforms, and networks. Currently, UNO is available on Linux, Solaris, and Windows for Java, C++ and OpenOffice.org Basic. As well, UNO is available through the component technology Microsoft COM for many other languages.

UNO is used to access OpenOffice.org, using its Application Programming Interface (API). The OpenOffice.org API is the comprehensive specification that describes the programmable features of OpenOffice.org.

2.2 Fields of Application for UNO

You can connect to a local or remote instance of OpenOffice.org from C++, Java and COM/DCOM. C++ and Java Desktop applications, Java servlets, Java Server Pages, JScript and VBScript, and languages, such as Delphi, Visual Basic and many others can use OpenOffice.org to work with Office documents.

It is possible to develop UNO Components in C++ or Java that can be instantiated by the office process and add new capabilities to OpenOffice.org. For example, you can write Chart Add-ins or Calc Add-ins, linguistic extensions, new file filters, database drivers. You can even write complete applications, such as a groupware client.

UNO components, as Java Beans, integrate with Java IDEs (Integrated Development Environment) to give easy access to OpenOffice.org. Currently, a set of such components is under development that will allow editing OpenOffice.org documents in Java Frames.

OpenOffice.org Basic cooperates with UNO, so that UNO programs can be directly written in OpenOffice.org. With this method, you supply your own office solutions and wizards based on an event-driven dialog environment.

The OpenOffice.org database engine and the data aware forms open another wide area of opportunities for database driven solutions.

2.3 Getting Started

A number of files and installation sets are required before beginning with the OpenOffice.org API.

2.3.1 Required Files

These files are required for any of the languages you use.

OpenOffice.org Installation
Install a copy of OpenOffice.org. The current version is OpenOffice.org 1.1.

You can download OpenOffice.org from *www.openoffice.org*. StarOffice can be obtained from Sun Microsystems or through your distributors.

Note: This book focuses on the current version.

API Reference
The OpenOffice.org API reference is part of the Software Development Kit and provides detailed information about OpenOffice.org objects. The latest version can be downloaded from the documents section at *api.openoffice.org*.

2.3.2 Installation Sets

The following installation sets are necessary to develop OpenOffice.org API applications with Java. This chapter describes how to set up a Java IDE for the OpenOffice.org API.

JDK 1.3.1
Java applications for OpenOffice.org 1.1 require the Java Development Kit 1.3.1 or later. Download and install a JDK from *java.sun.com*. To get all features Java 1.4.1_01 is required.

Java IDE
Download an Integrated Development Environment (IDE), such as NetBeans from *www.netbeans.org* or Forte for Java from Sun Microsystems. Other IDEs can be used, but NetBeans/Forte offers the best integration. The integration of OpenOffice.org with IDEs such as NetBeans is an ongoing effort. Check the files section of *api.openoffice.org* for the latest information about NetBeans and other IDEs.

OpenOffice.org Software Development Kit (SDK)
Obtain the OpenOffice.org Software Development Kit (SDK) from *www.openoffice.org*. It contains the build environment for the examples mentioned in this manual and reference documentation for the OpenOffice.org API, for the Java UNO runtime, and the C++ API. It also offers more example sources. By means of the SDK you can use GNU *make* to build and run the examples we mention here.

Unpack the SDK somewhere in your file system. The file *index.html* gives an overview of the SDK. For detailed instructions which compilers to use and how to set up your development environment, please refer to the SDK installation guide.

2.3.3 Configuration

Enable Java in OpenOffice.org

OpenOffice.org uses a Java Virtual Machine to instantiate components written in Java. You can easily tell the office which JVM to use: launch the *jvmsetup* executable from the programs folder under the OpenOffice.org, select an installed JRE or JDK and click **OK**. Close the OpenOffice.org including the Quickstarter in the taskbar and restart OpenOffice.org. Furthermore, open the **Tools - Options** dialog in OpenOffice.org, select the section **OpenOffice.org - Security** and make sure that the **Java enable** option is checked.

Use Java UNO class files

Next, the OpenOffice.org class files must be made known to the Java IDE. For NetBeans these Java UNO jar files must be mounted to a project. The following steps show how to create a new project and mount class files in NetBeans from version 3.4.1.

1. From the **Project** menu, select **Project Manager**. Click the **New** ... button in the **Project Manager** window to create a new project. NetBeans uses your new project as the current project.
2. Activate the NetBeans **Explorer** window—it should contain a **Filesystems** item (to display the NetBeans Explorer window, click **View - Explorer**). Open its context menu and select **Mount – Archive Files**, navigate to the folder *<OfficePath>/program/classes*, choose all jar files in that directory and click **Finish** to mount the OpenOffice.org jars in your project. As an alternative, you can also mount files using **File - Mount Filesystem**.
3. Finally you need a folder for the source files of your project. Choose **Mount – Local Directory** from the context menu of the **Filesystems** icon and use the file manager dialog to create a new folder somewhere in your file system. Select it without opening it and click **Finish** to add it to your project.

Make the office listen

Java uses a TCP/IP socket to talk to the office. For Java clients, OpenOffice.org must be told to listen for TCP/IP connections using a special connection url parameter. There are two ways to achieve this, you can make the office listen *always* or just *once*.

- To make the office listen whenever it is started, open the file *<OfficePath>/share/registry/data/org/openoffice/Setup.xcu* in an editor, and look for the element `<node oor:name="Office"/>`
  This element contains `<prop/>` elements. Insert the following `<prop/>` element on the same level as the existing elements:
  ```
  <prop oor:name="ooSetupConnectionURL">
    <value>socket,host=localhost,port=8100;urp;</value>
  </prop>
  ```
  This setting configures OpenOffice.org to provide a socket on port 8100, where it will serve connections through the UNO remote protocol (urp). If port 8100 is already in use on your

machine, it may be necessary to adjust the port number. Block port 8100 for connections from outside your network in your firewall. If you have a OpenOffice.org network installation, this setting will affect all users. To make only a particular user installation listen for connections create a file *<OfficePath>/user/registry/data/org/openoffice/Setup.xcu* with the same structure as the file above and add the element `<prop oor:name="ooSetupConnectionURL"/>` as shown above.

- An alternative is to launch the office in listening mode using command-line options. To do this, start it from the command-line:
  `<OfficePath>/program/soffice "-accept=socket,port=8100;urp;"`
  When you use this command-line option, the office will only listen during the current session, and running instances are affected as well.

Choose the procedure that suits your requirements and launch OpenOffice.org in listening mode now. Check if it is listening by calling *netstat -a* on the command-line. An output similar to the following shows that the office is listening:

```
TCP    <Hostname>:8100   <Fully qualified hostname>: 0 Listening
```

If the office is not listening, it probably was not started with the proper connection URL parameter. Check the *Setup.xcu* file or your command-line for typing errors and try again.



Note: In versions before OpenOffice.org 1.1, there are several differences.

The configuration setting that makes the office listen everytime is located elsewhere. Open the file *<OfficePath>/share/config/registry/instance/org/openoffice/Setup.xml* in an editor, and look for the element:

```
<ooSetupConnectionURL cfg:type="string"/>
```

Extend it with the following code:

```
<ooSetupConnectionURL cfg:type="string">
socket,port=8100;urp;
</ooSetupConnectionURL>
```

The commandline option -accept is ignored when there is a running instance of the office, including the quick starter and the online help. If you use it, make sure that no *soffice* process runs on your system.

Add the API Reference to your IDE

We recommend to add the API and the Java UNO reference to your Java IDE to get online help for the OpenOffice.org API and the Java UNO runtime. In NetBeans 3.4.1, follow these steps:

- Open your project and choose the **Tools – Javadoc Manager** menu. With the button **Add Folder...** add the folders *docs/common/ref* and *docs/java/ref* of your SDK installation to use the API and the Java UNO reference in your project.
- You can now use **Alt** + **F1** to view online help while the cursor is on a OpenOffice.org API or Java UNO identifier in the source editor window.

2.3.4 First Connection

Getting Connected

The following demonstrates how to write a small program that connects to the office and tells you if it was able to establish the connection or not. Start the Java IDE or source editor, and enter the following source code for the `FirstConnection` class.

To create and run the class in the NetBeans 3.4.1 IDE, use the following steps:

1. Add a main class to the project. In the NetBeans Explorer window, click the **Project <project_name>** tab, right click the **Project** item, select **Add New...** to display the **New Wizard**, open the **Java Classes** folder, highlight the template **Main,** and hit **Next**.
2. In the **Name** field, enter 'FirstConnection' as classname for the Main class and select the folder that contains your project files. The `FirstConnection` is added to the default package of your project. Click **Finish** to create the class.
3. Enter the source code shown below (FirstSteps/FirstConnection.java). Then select **Build - Execute** to test your first connection. Observe the **Output** window where NetBeans displays the result of your attempt to connect to the office.

```
import com.sun.star.bridge.XUnoUrlResolver;
import com.sun.star.uno.UnoRuntime;
import com.sun.star.uno.XComponentContext;
import com.sun.star.lang.XMultiComponentFactory;
import com.sun.star.beans.XPropertySet;

public class FirstConnection extends java.lang.Object {

    private XComponentContext xRemoteContext = null;
    private XMultiComponentFactory xRemoteServiceManager = null;

    public static void main(String[] args) {
        FirstConnection firstConnection1 = new FirstConnection();
        try {
            firstConnection1.useConnection();
        }
        catch (java.lang.Exception e) {
            e.printStackTrace();
        }
        finally {
            System.exit(0);
        }
    }

    protected void useConnection() throws java.lang.Exception {
        try {
            xRemoteServiceManager = this.getRemoteServiceManager(
                    "uno:socket,host=localhost,port=8100;urp;StarOffice.ServiceManager");
            String available = (null != xRemoteServiceManager ? "available" : "not available");
            System.out.println("remote ServiceManager is " + available);

            //
            // do something with the service manager...
            //

        }
         catch (com.sun.star.connection.NoConnectException e) {
                 System.err.println("No process listening on the resource");
                 e.printStackTrace();
                 throw e;
         }
          catch (com.sun.star.lang.DisposedException e) { //works from Patch 1
            xRemoteContext = null;
            throw e;
        }
    }

    protected XMultiComponentFactory getRemoteServiceManager(String unoUrl) throws java.lang.Exception {
        if (xRemoteContext == null) {
            // First step: create local component context, get local servicemanager and
            // ask it to create a UnoUrlResolver object with an XUnoUrlResolver interface
            XComponentContext xLocalContext =
```

```
                com.sun.star.comp.helper.Bootstrap.createInitialComponentContext(null);

            XMultiComponentFactory xLocalServiceManager = xLocalContext.getServiceManager();

            Object urlResolver  = xLocalServiceManager.createInstanceWithContext(
                "com.sun.star.bridge.UnoUrlResolver", xLocalContext );
            // query XUnoUrlResolver interface from urlResolver object
            XUnoUrlResolver xUnoUrlResolver = (XUnoUrlResolver) UnoRuntime.queryInterface(
                XUnoUrlResolver.class, urlResolver);

            // Second step: use xUrlResolver interface to import the remote StarOffice.ServiceManager,
            // retrieve its property DefaultContext and get the remote servicemanager
            Object initialObject = xUnoUrlResolver.resolve(unoUrl);
            XPropertySet xPropertySet = (XPropertySet)UnoRuntime.queryInterface(
                XPropertySet.class, initialObject);
            Object context = xPropertySet.getPropertyValue("DefaultContext");
            xRemoteContext = (XComponentContext)UnoRuntime.queryInterface(
                XComponentContext.class, context);
        }
        return xRemoteContext.getServiceManager();
    }
}
```

For an example that connects to the office with C++, see chapter *3.4.2 Professional UNO - UNO Language Bindings - UNO C++ Binding*. Accessing the office with OpenOffice.org Basic is described in 11.1 Basic and Dialogs - First Steps with OpenOffice.org Basic.

The next section describes what happens during the connection between a Java program and OpenOffice.org.

Service Managers

UNO introduces the concept of *service managers*, which can be considered as “factories” that create *services*. For now, it is sufficient to see services as UNO objects that can be used to perform specific tasks. Later on we will give a more precise definition for the term service.

For example, the following services are available:

com.sun.star.frame.Desktop
: maintains loaded documents: is used to load documents, to get the current document, and access all loaded documents

com.sun.star.configuration.ConfigurationProvider
: yields access to the OpenOffice.org configuration, for instance the settings in the **Tools - Options** dialog

com.sun.star.sdb.DatabaseContext
: holds databases registered with OpenOffice.org

com.sun.star.system.SystemShellExecute
: executes system commands or documents registered for an application on the current platform

com.sun.star.text.GlobalSettings
: manages global view and print settings for text documents

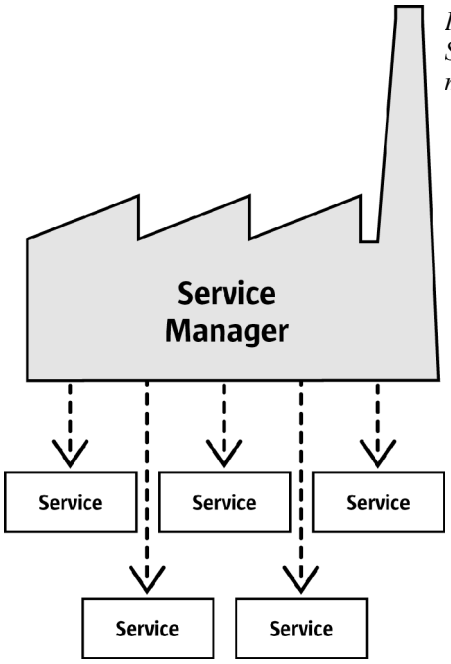


Illustration 1: Service manager

A service always exists in a *component context*, which consists of the service manager that created the service and other data to be used by the service.

The `FirstConnection` class above is considered a client of the OpenOffice.org process, OpenOffice.org is the server in this respect. Both, client and server, have their own component context and their own service manager. The client service manager creates UNO objects in the client process and the server service manager does the same in the server process. This is necessary because only UNO objects created by a service manager can talk to each other across process boundaries. Now, to work with data located on the server side, the client needs to get the server's service manager.

The method `getRemoteServiceManager()` of our `FirstConnection` class uses two steps to achieve this:

First, it gets a local UNO component context from the class `com.sun.star.comp.helper.Bootstrap`. This local component context contains a small service manager that knows how to create the services that are necessary to talk to other component contexts. One such service is a `com.sun.star.bridge.UnoUrlResolver`, so it asks the local service manager to create this service.

Next, it uses the `UnoUrlResolver` object to get the component context together with the service manager from the server-side. At this stage you should not worry about the `queryInterface()` calls—they will be explained later. Now the client holds a reference to the remote service manager.

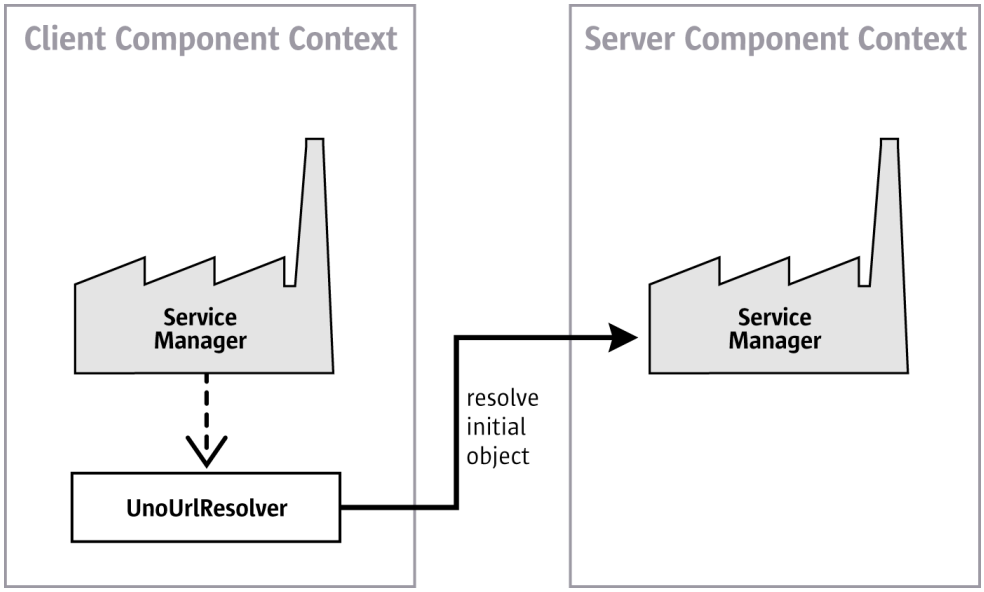


Illustration 2: UnoUrlResolver gets Remote ServiceManager

For a thorough description of the objects used here, refer to the chapter *3.3.1 Professional UNO - UNO Concepts - UNO Interprocess Connections*.

Failed Connections

A remote connection can fail under certain conditions:

- Client programs should be able to detect errors. For instance, sometimes the bridge might become unavailable. Simple clients that connect to the office, perform a certain task and exit afterwards should stop their work and inform the user if an error occurred.
- Clients that are supposed to run over a long period of time should not assume that a reference to an initial object will be valid over the whole runtime of the client. The client should resume even if the connection goes down for some reason and comes back later on. When the connection fails, a robust, long running client should stop the current work, inform the user that the connection is not available and release the references to the remote process. When the user tries to repeat the last action, the client should try to rebuild the connection. Do not force the user to restart your program just because the connection was temporarily unavailable.

When the bridge has become unavailable and access is tried, it throws a `com.sun.star.lang.DisposedException`. Whenever you access remote references in your program, catch this Exception in such a way that you set your remote references to null and inform the user accordingly. If your client is designed to run for a longer period of time, be prepared to get new remote references when you find that they are currently null.

Another way is to register a listener at the *remote bridge* that underlies the `UnoUrlResolver`. OpenOffice.org allows to listen for a "bridge disposed" event at the remote bridge so that you can release invalid references—inform the user what has happened or throw a suitable exception if necessary. To do this, you must manually create a bridge and register a listener at the bridge. A connection created by `UnoUrlResolver` simply throws a `java.lang.RuntimeException` whenever you try to use a reference that no longer works because of a connection failure. The chapter *3.3.1 Professional UNO - UNO Concepts - UNO Interprocess Connections* shows how to write such a connection aware client.

2.4 How to get Objects in OpenOffice.org

An *object* in our context is an instance of an implemented class that has methods you can call. Objects are required to do something with OpenOffice.org. But where do you obtain them?

New objects

In general, new objects or objects which are necessary for a first access are created by *service managers* in OpenOffice.org. In the `FirstConnection` example, the local service manager created a `UnoUrlResolver` object:

```
Object urlResolver  = xLocalServiceManager.createInstanceWithContext(
                "com.sun.star.bridge.UnoUrlResolver", xLocalContext );
```

The remote service manager works exactly like the local service manager. The remote service manager creates the remote `Desktop` object, which handles application windows and loaded documents in OpenOffice.org:

```
Object desktop = xRemoteServiceManager.createInstanceWithContext(
                 "com.sun.star.frame.Desktop", xRemoteContext);
```

Document objects

Document objects represent the files that are opened with OpenOffice.org. They are created by the `Desktop` object, which has a `loadComponentFromURL()` method for this purpose.

Objects that are provided by other objects

Objects can hand out other objects. There are two cases:

- Features which are designed to be an integral part of the object that provides the feature can be obtained by get methods in the OpenOffice.org API. It is common to get an object from a get method. For instance, `getSheets()` is required for every Calc document, `getText()` is essential for every Writer Document and `getDrawpages()` is an essential part of every Draw document. After loading a document, these methods are used to get the Sheets, Text and Drawpages object of the corresponding document. Object-specific get methods are an important technique to get objects.

- Features which are not considered integral for the architecture of an object are accessible through a set of universal methods. In the OpenOffice.org API, these features are called properties, and generic methods are used, such as `getPropertyValue(String propertyName)` to access them. In some cases such a non-integral feature is provided as an object, therefore the method `getPropertyValue()` can be another source for objects. For instance, page styles for spreadsheets have the properties `"RightPageHeaderContent"` and `"LeftPageHeaderContent"`, that contain objects for the page header sections of a spreadsheet document. The generic `getPropertyValue()` method can sometimes provide an object you need.

Sets of objects

Objects can be elements in a set of similar objects. In sets, to access an object you need to know how to get a particular element from the set. The OpenOffice.org API allows four ways to provide an element in a set. The first three ways are objects with element access methods that allow access by name, index, or enumeration. The fourth way is a sequence of elements which has no access methods but can be used as an array directly. How these sets of elements are used will be discussed later.

The designer of an object decides which of those opportunities to offer, based on special conditions of the object, such as how it performs remotely or which access methods best work with implementation.

2.5 Working with Objects

Working with OpenOffice.org API objects involves the following:

- First we will learn why UNO describes objects as services, consisting of interfaces and properties and we will get acquainted with UNO's way to use interfaces and properties.
- After that, we will work with a OpenOffice.org document for the first time, and give some hints for the usage of the most common types in OpenOffice.org API.
- Finally we will introduce the common interfaces that allow you to work with text, tables and drawings across all OpenOffice.org document types.

2.5.1 Services

In the OpenOffice.org API, objects are called *services*. However, objects and services are not the same thing. Services are abstract specifications for objects. All UNO objects have to follow a service specification and have to support at least one service. A UNO object is called a service, because it fulfills a service specification.

A service describes an object by combining *interfaces* and *properties* into an abstract object specification. Do not get confused by the meanings the word service has in other contexts. In UNO, a service is precisely this: a composition of interfaces and properties.

An *interface* is a set of methods that together define one single aspect of a service. For instance, the `com.sun.star.view.XPrintable` interface prescribes the methods `print()`, `getPrinter()` and `setPrinter()`.

A *property* is a feature of a service which is not considered an integral or structural part of the service and therefore is handled through generic `getPropertyValue()`/`setPropertyValue()` methods instead of specialized get methods, such as `getPrinter()`. An object containing properties only has to support the `com.sun.star.beans.XPropertySet` interface to be prepared to handle all kinds of properties. Typical examples are properties for character or paragraph formatting. With properties, you can set multiple features of an object through a single call to `setPropertyValues()`, which greatly improves the remote performance. For instance, paragraphs support the `setPropertyValues()` method through their `com.sun.star.beans.XMultiPropertySet` interface.

The concept of services was introduced for the following reasons:

Services separate specification from implementation
: The specification of a service is *abstract*, that is, it does not define how objects supporting a certain functionality do this *internally*. Through the abstract specification of the OpenOffice.org API, it is possible to pull the implementation out from under the API and install a different implementation if required.

Service names allow to create instances by specification name, not by class names
: In Java or C++ you use the `new` operator to create a class instance. This approach is restricted: the class you get is hard-coded. You cannot later on exchange it by another class without editing the code. The concept of services solves this. The central object factory in OpenOffice.org, the global service manager, is asked to create an object that can be used for a certain purpose without defining its internal implementation. This is possible, because a service can be ordered from the factory by its *service name* and the factory decides which service implementation it returns. Which implementation you get makes no difference, you only use the well-defined interfaces and properties of the service.

Services make fine-grained interfaces manageable

Abstract interfaces are more reusable, if they are fine-grained, i.e. if they are small and describe only one aspect of an object, not several aspects. But then you need many of them to describe a useful object. Services allow to have fine-grained interfaces on the one hand *and* to manage them easily by forging them into a service. Since it is quite probable that objects in an office environment will share many aspects, this fine granularity allows the interfaces to be reused and thus to get objects that behave consistently. For instance, it was possible to realize a unified way to handle text, no matter if you are dealing with body text, text frames, header or footer text, footnotes, table cells or text in drawing shapes. It was not necessary to define separate interfaces for all of these purposes.

Services handle a large number of non-structural properties

If you have only interfaces to specify objects, you need many get and set methods to handle all the qualities of office documents. Moreover, once you define them, they become a hard part of the structure of an object, which makes it difficult to reuse the specification elsewhere. With properties, a multitude of qualities can be specified that are no structural parts of the objects, and instead of calling many get and set methods, properties in a UNO service can be manipulated at once by a single method call, if necessary.

Let us consider the service `com.sun.star.text.TextDocument` in UML notation. The UML chart shown in Illustration 3 depicts the mandatory interfaces of a `TextDocument` service. These interfaces express the basic aspects of a text document in OpenOffice.org. It contains text, it is searchable and refreshable. It is a model with URL and controller, and it is modifiable, printable and storable. The UML chart shows how this is specified in the API.

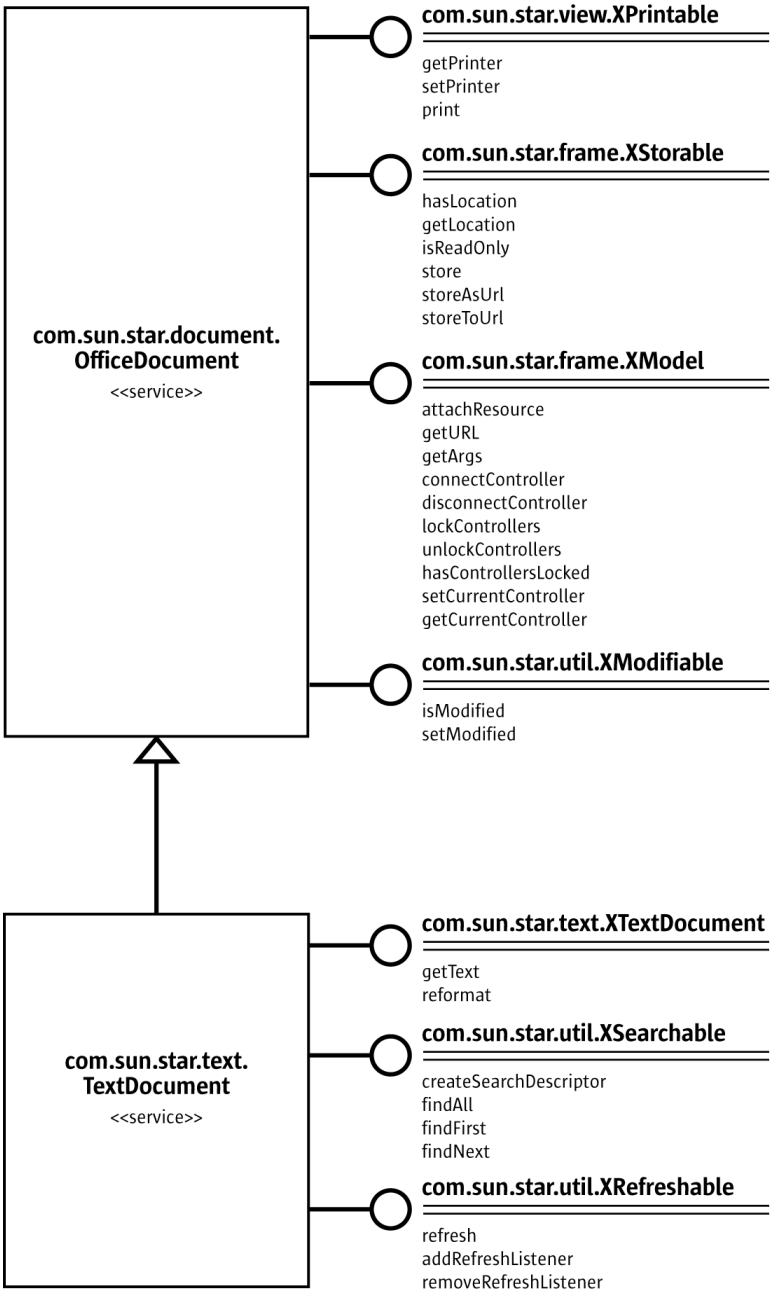


Illustration 3: Text Document

On the left of Illustration 3, the services `com.sun.star.text.TextDocument` and `com.sun.star.document.OfficeDocument` are shown. Every `TextDocument` must include these services by definition.

On the right of Illustration 3, you find the interfaces, that the services must export. Their method compartments list the methods contained in the various interfaces. In the OpenOffice.org API, all interface names have to start with an X to be distinguishable from other object names.

Every `TextDocument` must support three interfaces: `XTextDocument`, `XSearchable`, and `XRefreshable`. In addition, because a `TextDocument` is always an `OfficeDocument`, it must also export the interfaces `XPrintable`, `XStorable`, `XModifiable` and `XModel`. The methods contained in these interfaces cover these aspects: printing, storing, modification and model handling.

Note that the interfaces shown in Illustration 2.2 are only the mandatory interfaces of a `TextDocument`. A `TextDocument` has optional properties and interfaces, among them the properties `CharacterCount`, `ParagraphCount` and `WordCount` and the `XPropertySet` interface which must be supported if properties are present at all. The current implementation of the `TextDocument` service in OpenOffice.org does not only support these interfaces, but all optional interfaces as well. The usage of a `TextDocument` is described thoroughly in *7 Text Documents*.

Using Interfaces

The fact that every UNO object must be accessed through its interfaces and properties has an effect in languages like Java and C++, where the compiler needs the correct type of an object reference before you can call a method from it. In Java or C++, you normally just cast an object before you access an interface it implements. When working with UNO objects this is different: You must ask the UNO environment to get the appropriate reference for you whenever you want to access methods of an interface which your object supports, but your compiler does not yet know about. Only then you can cast it safely.

The Java UNO environment has a method `queryInterface()` for this purpose. It looks complicated at first sight, but once you understand that `queryInterface()` is about safe casting of UNO types across process boundaries, you will soon get used to it. Remember how we created a UnoUrlResolver and afterwards had to call `queryInterface()` in our FirstConnection class:

```
Object urlResolver  = xLocalServiceManager.createInstanceWithContext(
                "com.sun.star.bridge.UnoUrlResolver", xLocalContext );

// query XUnoUrlResolver interface from urlResolver object
XUnoUrlResolver xUnoUrlResolver = (XUnoUrlResolver) UnoRuntime.queryInterface(
                XUnoUrlResolver.class, urlResolver );
```

We asked the local service manager to create a `com.sun.star.bridge.UnoUrlResolver` using its factory method `createInstanceWithContext()`. This method is defined to return a Java Object type, which should not surprise you—after all the factory must be able to return any type:

```
java.lang.Object createInstanceWithContext(String serviceName, XComponentContext context)
```

The object we receive is a `com.sun.star.bridge.UnoUrlResolver` service. Below you find its specification in UML notation. The service UnoUrlResolver has no properties and it supports one interface `com.sun.star.bridge.XUnoUrlResolver` with one method, namely `resolve()`:

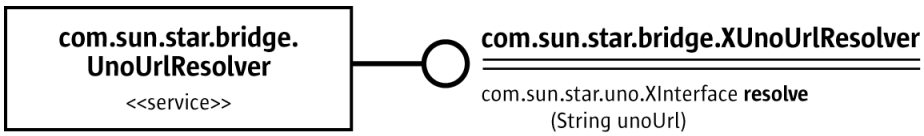


Illustration 4: UnoUrlResolver

The point is, while *we* know that the object we ordered at the factory is a UnoUrlResolver and exports the interface XUnoUrlResolver, the compiler does *not*. Therefore, we have to use the UNO runtime environment to ask or *query* for the interface `XUnoUrlResolver`, since we want to use the `resolve()` method on this interface. The method `queryInterface()` makes sure we get a reference that can be cast to the needed interface type, no matter if the target object is a local or a remote object. There are two `queryInterface` definitions in the Java UNO language binding:

```
java.lang.Object UnoRuntime.queryInterface(java.lang.Class targetInterface, Object sourceObject)
java.lang.Object UnoRuntime.queryInterface(com.sun.star.uno.Type targetInterface, Object sourceObject)
```

Since `UnoRuntime.queryInterface()` is specified to return a java.lang.Object just like the factory method `createInstanceWithContext()`, we still must explicitly cast our interface reference to the needed type. The difference is that after `queryInterface()` we can safely cast the object to our interface type and, most important, that the reference will now work even with an object in another process. Here is the `queryInterface()` call, explained step by step:

```
XUnoUrlResolver xUnoUrlResolver = (XUnoUrlResolver) UnoRuntime.queryInterface(
          XUnoUrlResolver.class, urlResolver );
```

`XUnoUrlResolver` is the interface we want to use, so we define a `XUnoUrlResolver` variable named `xUnoUrlResolver` (lower x) to store the interface we expect from `queryInterface`.

Then we query our `urlResolver` object for the `XUnoUrlResolver` interface, passing in `XUnoUrlResolver.class` as target interface and `urlResolver` as source object. Finally we cast the outcome to `XUnoUrlResolver` and assign the resulting reference to our variable `xUnoUrlResolver`.

If the source object does not support the interface we are querying for, `queryInterface()` will return `null`.

In Java, this call to `queryInterface()` is necessary whenever you have a reference to an object which is known to support an interface that you need, but you do not have the proper reference type yet. Fortunately, you are not only allowed to `queryInterface()` from `java.lang.Object` source types, but you may also query an interface from another interface reference, like this:

```
// loading a blank spreadsheet document gives us its XComponent interface:
XComponent xComponent = xComponentLoader.loadComponentFromURL(
                "private:factory/scalc", "_blank", 0, loadProps);

// now we query the interface XSpreadsheetDocument from xComponent
XSpreadsheetDocument xSpreadsheetDocument = (XSpreadsheetDocument)UnoRuntime.queryInterface(
               XSpreadsheetDocument.class, xComponent);
```

Furthermore, if a method is defined in such a way that it already returns an interface type, you do not need to query the interface, but you can use its methods right away. In the snippet above, the method `loadComponentFromURL` is specified to return an `com.sun.star.lang.XComponent` interface, so you may call the `XComponent` methods `addEventListener()` and `removeEventListener()` directly at the `xComponent` variable, if you want to be notified that the document is being closed.



It is possible that future versions of the Java UNO language binding will no longer need explicit queries for interfaces.

The corresponding step in C++ is done by a `Reference<>` template that takes the source instance as parameter:

```
// instantiate a sample service with the servicemanager.
Reference< XInterface > rInstance =
        rServiceManager->createInstanceWithContext(
                OUString::createFromAscii("com.sun.star.bridge.UnoUrlResolver" ),
                rComponentContext );

// Query for the XUnoUrlResolver interface
Reference< XUnoUrlResolver > rResolver( rInstance, UNO_QUERY );
```

In OpenOffice.org Basic, querying for interfaces is not necessary, the Basic runtime engine takes care about that internally.

Using Properties

A service must offer its properties through interfaces that allow you to work with properties. The most basic form of these interfaces is the interface `com.sun.star.beans.XPropertySet`. There are other interfaces for properties, such as `com.sun.star.beans.XMultiPropertySet`, that gets and sets a multitude of properties with a single method call. The `XPropertySet` is always supported when properties are present in a service.

In `XPropertySet`, two methods carry out the property access, which are defined in Java as follows:

```
void setPropertyValue(String propertyName, Object propertyValue)
Object getPropertyValue(String propertyName)
```

In the FirstConnection example, the `XPropertySet` interface was used to get the remote component context from the initial object. The initial object was a `StarOffice.ServiceManager` and therefore had a property `DefaultContext` which contained the remote component context. The following code explains how this property was retrieved and queried its `com.sun.star.uno.XComponentContext` interface:

```
// query the XPropertySet interface from the initial object, which is a StarOffice.ServiceManager
XPropertySet xPropertySet = (XPropertySet)UnoRuntime.queryInterface(
                XPropertySet.class, initialObject);

// get the property DefaultContext
Object context = xPropertySet.getPropertyValue("DefaultContext");

// query XComponentContext from the context object, we want to call XComponentContext.getServiceManager
xRemoteContext = (XComponentContext)UnoRuntime.queryInterface(
                XComponentContext.class, context);
```

You are now ready to start working with a OpenOffice.org document.

2.5.2 Example: Working with a Spreadsheet Document

In this example, we will ask the remote service manager to give us the remote `Desktop` object and use its `loadComponentFromUrl()` method to create a new spreadsheet document. From the document we get its sheets container where we insert and access a new sheet by name. In the new sheet, we enter values into A1 and A2 and summarize them in A3. The cell style of the summarizing cell gets the cell style Result, so that it appears in italics, bold and underlined. Finally, we make our new sheet the active sheet, so that the user can see it.

Add these import lines to the FirstConnection example above:
(FirstSteps/FirstLoadComponent.java)

```
import com.sun.star.beans.PropertyValue;
import com.sun.star.lang.XComponent;
import com.sun.star.sheet.XSpreadsheetDocument;
import com.sun.star.sheet.XSpreadsheets;
import com.sun.star.sheet.XSpreadsheet;
import com.sun.star.sheet.XSpreadsheetView;
import com.sun.star.table.XCell;
import com.sun.star.frame.XModel;
import com.sun.star.frame.XController;
import com.sun.star.frame.XComponentLoader;
```

Edit the useConnection method as follows:

```
protected void useConnection() throws java.lang.Exception {
    try {

        xRemoteServiceManager = this.getRemoteServiceManager(
                "uno:socket,host=localhost,port=8100;urp;StarOffice.ServiceManager");

        // get the Desktop, we need its XComponentLoader interface to load a new document
        Object desktop = xRemoteServiceManager.createInstanceWithContext(
            "com.sun.star.frame.Desktop", xRemoteContext);

        // query the XComponentLoader interface from the desktop
        XComponentLoader xComponentLoader = (XComponentLoader)UnoRuntime.queryInterface(
            XComponentLoader.class, desktop);

        // create empty array of PropertyValue structs, needed for loadComponentFromURL
        PropertyValue[] loadProps = new PropertyValue[0];

        // load new calc file
        XComponent xSpreadsheetComponent = xComponentLoader.loadComponentFromURL(
            "private:factory/scalc", "_blank", 0, loadProps);

        // query its XSpreadsheetDocument interface, we want to use getSheets()
        XSpreadsheetDocument xSpreadsheetDocument = (XSpreadsheetDocument)UnoRuntime.queryInterface(
            XSpreadsheetDocument.class, xSpreadsheetComponent);


        // use getSheets to get spreadsheets container
        XSpreadsheets xSpreadsheets = xSpreadsheetDocument.getSheets();
```

```
        //insert new sheet at position 0 and get it by name, then query its XSpreadsheet interface
        xSpreadsheets.insertNewByName("MySheet", (short)0);
        Object sheet = xSpreadsheets.getByName("MySheet");
        XSpreadsheet xSpreadsheet = (XSpreadsheet)UnoRuntime.queryInterface(
            XSpreadsheet.class, sheet);

        // use XSpreadsheet interface to get the cell A1 at position 0,0 and enter 21 as value
        XCell xCell = xSpreadsheet.getCellByPosition(0, 0);
        xCell.setValue(21);

        // enter another value into the cell A2 at position 0,1
        xCell = xSpreadsheet.getCellByPosition(0, 1);
        xCell.setValue(21);

        // sum up the two cells
        xCell = xSpreadsheet.getCellByPosition(0, 2);
        xCell.setFormula("=sum(A1:A2)");

        // we want to access the cell property CellStyle, so query the cell's XPropertySet interface
        XPropertySet xCellProps = (XPropertySet)UnoRuntime.queryInterface(
            XPropertySet.class, xCell);

        // assign the cell style "Result" to our formula, which is available out of the box
        xCellProps.setPropertyValue("CellStyle", "Result");

        // we want to make our new sheet the current sheet, so we need to ask the model
        // for the controller: first query the XModel interface from our spreadsheet component
        XModel xSpreadsheetModel = (XModel)UnoRuntime.queryInterface(
           XModel.class, xSpreadsheetComponent);

        // then get the current controller from the model
        XController xSpreadsheetController = xSpreadsheetModel.getCurrentController();

        // get the XSpreadsheetView interface from the controller, we want to call its method
        // setActiveSheet
        XSpreadsheetView xSpreadsheetView = (XSpreadsheetView)UnoRuntime.queryInterface(
           XSpreadsheetView.class, xSpreadsheetController);

        // make our newly inserted sheet the active sheet using setActiveSheet
        xSpreadsheetView.setActiveSheet(xSpreadsheet);
    }
    catch( com.sun.star.lang.DisposedException e ) { //works from Patch 1
        xRemoteContext = null;
        throw e;
    }
}
```

Alternatively, you can add *FirstLoadComponent.java* from the samples directory to your current project, it contains the changes shown above.

2.5.3 Common Types

Until now, literals and common Java types for method parameters and return values have been used as if the OpenOffice.org API was made for Java. However, it is important to understand that the OpenOffice.org API is designed to be language independent and therefore has its own internal types which have to be mapped to the proper types for your language environment. The type mappings are briefly described in this section.Refer to *3 Professional UNO* for detailed information about type mappings.

Simple Types

Simple types occur in structs, method return values or parameters. The following table shows the simple types in UNO and, if available, their exact mappings to Java, C++, OpenOffice.org and Basic types.

UNO	Type description	Java	C++	Basic
`char`	16-bit unicode character type	`char`	`sal_Unicode`	-

UNO	Type description	Java	C++	Basic
boolean	boolean type; true and false	boolean	sal_Bool	Boolean
byte	8-bit ordinal type	byte	sal_Int8	Integer
short	signed 16-bit ordinal type	short	sal_Int16	Integer
unsigned short	unsigned 16-bit ordinal type	-	sal_uInt16	-
long	signed 32-bit ordinal type	int	sal_Int32	Long
unsigned long	unsigned 32-bit type	-	sal_uInt32	-
hyper	signed 64-bit ordinal type	long	sal_Int64	-
unsigned hyper	unsigned 64-bit ordinal type	-	sal_uInt64	-
float	processor dependent float	float	float (IEEE float)	Single
double	processor dependent double	double	double (IEEE double)	Double

There are special conditions for types that do not have an exact mapping in this table. Check for details about these types in the corresponding sections about type mappings in *3.4 Professional UNO - UNO Language Bindings*.

Strings

UNO considers strings to be simple types, but since they need special treatment in some environments, we discuss them separately here.

UNO	Description	Java	C++	Basic
string	string of 16-bit unicode characters	java.lang.String	::rtl::OUString	String

In Java, use UNO strings as if they were native `java.lang.String` objects.

In C++, strings must be converted to UNO unicode strings by means of SAL conversion functions, usually the function `createFromAscii()` in the ::rtl::OUString class:

```
//C++
static OUString createFromAscii( const sal_Char * value ) throw();
```

In Basic, Basic strings are mapped to UNO strings transparently.

Enum Types and Groups of Constants

The OpenOffice.org API uses many enumeration types (called enums) and groups of constants (called constant groups). *Enums* are used to list every plausible value in a certain context. The *constant groups* define possible values for properties, parameters, return values and struct members.

For example, there is an enum `com.sun.star.table.CellVertJustify` that describes the possible values for the vertical adjustment of table cell content. The vertical adjustment of table cells is determined by their property `com.sun.star.table.CellProperties:VertJustify`. The possible values for this property are, according to `CellVertJustify`, the values `STANDARD`, `TOP`, `CENTER` and `BOTTOM`.

```
// adjust a cell content to the upper cell border
// The service com.sun.star.table.Cell includes the service com.sun.star.table.CellProperties
// and therefore has a property VertJustify that controls the vertical cell adjustment
// we have to use the XPropertySet interface of our Cell to set it

xCellProps.setPropertyValue("VertJustify", com.sun.star.table.CellVertJustify.TOP);
```

OpenOffice.org Basic understands enumeration types and constant groups. Their usage is straightforward:

```
'OpenOffice.org Basic
oCellProps.VertJustify = com.sun.star.table.CellVertJustify.TOP
```

In C++ enums and constant groups are used with the scope operator ::

```
//C++
rCellProps->setPropertyValue(OUString::createFromAscii( "VertJustify" ),
        ::com::sun::star::table::CellVertJustify.TOP);
```

2.5.4 Struct

Structs in the OpenOffice.org API are used to create compounds of every other UNO type. They correspond to C structs or Java classes consisting of public member variables only.

While structs do not encapsulate data, they are easier to transport as a whole, instead of marshalling `get()` and `set()` calls back and forth. In particular, this has advantages for remote communication.

You gain access to struct members through the . (dot) operator as in

```
aProperty.Name = "ReadOnly";
```

In Java, C++ und OpenOffice.org Basic, the keyword `new` instantiates structs. In OLE automation, use `com.sun.star.reflection.CoreReflection` to get a UNO struct. Do not use the service manager to create structs.

```
//In Java:
com.sun.star.beans.PropertyValue aProperty = new com.sun.star.beans.PropertyValue();

'In StarBasic
Dim aProperty as new com.sun.star.beans.PropertyValue
```

2.5.5 Any

The OpenOffice.org API frequently uses an `any` type, which is the counterpart of the `Variant` type known from other environments. The `any` type holds one arbitrary UNO type. The `any` type is especially used in generic UNO interfaces.

Examples for the occurrence of `any` are the method parameters and return values of the following, frequently used methods:

Interface	returning an any type	taking an any type
`XPropertySet`	`any` **`getPropertyValue`**`(string propertyName)`	`void` **`setPropertyValue`**`(any value)`

Interface	returning an any type	taking an any type	
XNameContainer	any **getByName**(string name)	void **replaceByName** (string name, any element)	void **insertByName** (string name, any element)
XIndexContainer	any **getByIndex**(long index)	void **replaceByIndex** (long index, any element)	void **insertByIndex** (long index, any element)
XEnumeration	any **nextElement**()	-	

Furthermore, the `any` type occurs in the `com.sun.star.beans.PropertyValue` struct.

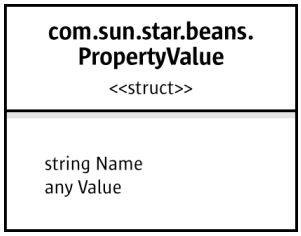


Illustration 5: PropertyValue

This `struct` has two member variables, `Name` and `Value`, and is ubiquitous in sets of `PropertyValue` structs, where every `PropertyValue` is a name-value pair that describes a property by name and value. If you need to set the value of such a `PropertyValue` struct, you must assign an `any` type, and you must be able to interpret the contained `any`, if you are reading from a `PropertyValue`. It depends on your language how this is done.

In Java, the `any` type is wrapped in a `java.lang.Object`. There are two simple rules to follow:

When you are supposed to *pass in* an any type, always pass in a `java.lang.Object` or a Java UNO object.

For instance, if you use `setPropertyValue()` to set a property that has a fundamental type in the target object, you must pass in a `java.lang.Object` for the new value. If the new value is a fundamental type in Java, create the corresponding `Object` type for the fundamental type:

```
xCellProps.setPropertyValue("CharWeight", new Double(200.0));
```

Another example would be a `PropertyValue` struct you want to use for `loadComponentFromURL`:

```
com.sun.star.beans.PropertyValue aProperty = new com.sun.star.beans.PropertyValue();
aProperty.Name = "ReadOnly";
aProperty.Value = new Boolean(true);
```

When you *receive* an `any` type, there are three different ways to evaluate it, depending on the UNO type you expect. If the incoming object has interfaces, use `queryInterface()` against it. If the incoming object is a struct, cast the incoming object to a Java UNO struct. If the incoming object is a simple type, use the `com.sun.star.uno.AnyConverter`.

The following is an example of a cast:

```
// the com.sun.star.table.TableBorder property that can be found in tables is a struct
// simply cast the property to the correct UNO struct type
com.sun.star.table.TableBorder bord = (TableBorder)xTableProps.getPropertyValue("TableBorder");

// now you can access the struct member fields
com.sun.star.table.BorderLine theLine = bord.TopLine;
int col = theLine.Color;
System.out.println(col);
```

The `AnyConverter` requires a closer look. For instance, if you want to get a property which contains a fundamental type, you must be aware that `getPropertyValue()` returns a java.lang.Object containing your fundamental type wrapped in an `any` type. The `com.sun.star.uno.AnyConverter` is a converter for such objects. Actually it can do more than

just conversion, you can find its specification in the Java UNO reference. The following list sums up the conversion functions in the `AnyConverter`:

```
static java.lang.Object toArray(java.lang.Object object)
static boolean toBoolean(java.lang.Object object)
static byte toByte(java.lang.Object object)
static char toChar(java.lang.Object object)
static double toDouble(java.lang.Object object)
static float toFloat(java.lang.Object object)
static int toInt(java.lang.Object object)
static long toLong(java.lang.Object object)
static java.lang.Object toObject(Class clazz, java.lang.Object object)
static java.lang.Object toObject(Type type, java.lang.Object object)
static short toShort(java.lang.Object object)
static java.lang.String toString(java.lang.Object object)
static Type toType(java.lang.Object object)
static int toUnsignedInt(java.lang.Object object)
static long toUnsignedLong(java.lang.Object object)
static short toUnsignedShort(java.lang.Object object)
```

Its usage is straightforward:

```
import com.sun.star.uno.AnyConverter;
long cellColor = AnyConverter.toLong(xCellProps.getPropertyValue("CharColor"));
```

In OpenOffice.org Basic, an `any` type becomes a Variant:

```
'OpenOffice.org Basic
Dim cellColor as Variant
cellColor = oCellProps.CharColor
```

In C++, there are special operators for the `any type`:

```
//C++ has >>= and <<= for Any (the pointed brackets are always left)
sal_Int32 cellColor;
Any any;
any = rCellProps->getPropertyValue(OUString::createFromAscii( "CharColor" ));
// extract the value from any
any >>= cellColor;
```

2.5.6 Sequence

A sequence is a set of UNO types with a variable number of elements that can be accessed directly without element access methods. Sequences map to arrays in most current language bindings. Although these sets in UNO are often implemented as objects with element access methods, there is also the sequence type, to be used where remote performance matters. Sequences are always written with pointed brackets in the API reference:

```
// a sequence of strings is notated as follows in the API reference
sequence < string > aStringSequence;
```

In Java, you treat sequences as arrays. Empty arrays are created using `new` and assigning a length of null. Furthermore, keep in mind that you only create an array of references when creating an array of Java objects, the actual objects are not allocated. Therefore, you must use `new` to create the array itself, then you must again use `new` for every single object and assign the new objects to the array.

An empty sequence of `PropertyValue` structs is frequently needed for `loadcomponentFromURL`:

```
// create an empty array of PropertyValue structs for loadComponentFromURL
PropertyValue[] emptyProps = new PropertyValue[0];
```

A sequence of `PropertyValue` structs is needed to use loading parameters with `loadComponent-FromURL()`. The possible parameter values for `loadComponentFromURL()` and the `com.sun.star.frame.XStorable` interface can be found in the service `com.sun.star.document.MediaDescriptor`.

```
// create an array with one PropertyValue struct for loadComponentFromURL, it contains references only
PropertyValue[] loadProps = new PropertyValue[1];
```

```
// instantiate PropertyValue struct and set its member fields
PropertyValue asTemplate = new PropertyValue();
asTemplate.Name = "AsTemplate";
asTemplate.Value = new Boolean(true);

// assign PropertyValue struct to first element in our array of references to PropertyValue structs
loadProps[0] = asTemplate;

// load calc file as template
XComponent xSpreadsheetComponent = xComponentLoader.loadComponentFromURL(
    "file:///X:/Office60/share/samples/english/spreadsheets/OfficeSharingAssoc.sxc",
    "_blank", 0, loadProps);
```

In OpenOffice.org Basic, a simple `Dim` creates an empty array.

```
'OpenOffice.org Basic
Dim loadProps()  'empty array
```

A sequence of structs is created using `new` together with `Dim`.

```
'OpenOffice.org Basic
Dim loadProps(0) as new com.sun.star.beans.PropertyValue 'one PropertyValue
```

In C++, there is a template for sequences. An empty sequence can be created by omitting the number of elements required.

```
//C++
Sequence < ::com::sun::star::beans::PropertyValue > loadProperties; // empty sequence
```

If you pass a number of elements, you get an array of the required type.

```
//C++
Sequence< ::com::sun::star::beans::PropertyValue > loadProps( 1 );
// the structs are default constructed
loadProps[0].Name = OUString::createFromAscii( "AsTemplate" );
loadProps[0].Handle <<= true;

Reference < XComponent > rComponent = rComponentLoader->loadComponentFromURL(
        OUString::createFromAscii("private:factory/swriter"),
        OUString::createFromAscii("_blank"),
        0,
        loadProps);
```

2.5.7 Element Access

We have already seen in section *2.4 First Steps - How to get Objects in OpenOffice.org* that sets of objects can also be provided through element access methods. The three most important kinds of element access interfaces are `com.sun.star.container.XNameContainer`, `[com.sun.star.container.XIndexContainer]` and `com.sun.star.container.XEnumeration`.

The three element access interfaces are examples of how the fine-grained interfaces of the OpenOffice.org API allow consistent object design.

All three interfaces inherit from `XElementAccess` , i.e. they include the methods:

```
type getElementType()
boolean hasElements()
```

to find out basic information about the set of elements. The method `hasElements()` answers the question if a set contains elements at all, and which type a set contains. In Java and C++, you can get information about a UNO type through `com.sun.star.uno.Type`, cf. the Java UNO and the C++ UNO reference.

The com.sun.star.container.XIndexContainer and com.sun.star.container.XNameContainer interface have a parallel design. Consider both interfaces in UML notation.

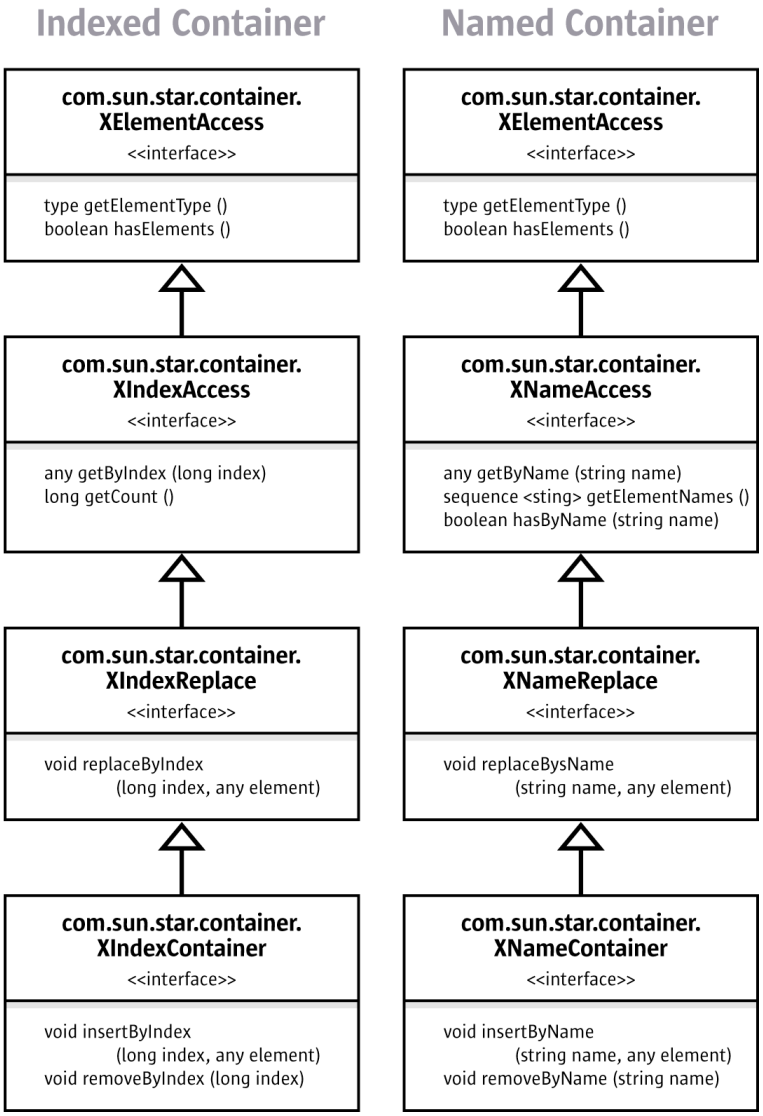


Illustration 6: Indexed and Named Container

The XIndexAccess/XNameAccess interfaces are about *getting* an element. The XIndexReplace/XNameReplace interfaces allow you to *replace existing* elements without changing the number of elements in the set, whereas the XIndexContainer/XNameContainer interfaces allow you to *increase and decrease the number of elements* by inserting and removing elements.

Many sets of named or indexed objects do not support the whole inheritance hierarchy of XIndex-Container or XNameContainer, because the capabilities added by every subclass are not always logical for any set of elements.

The XEumerationAccess interface works differently from named and indexed containers below the XElementAccess interface. XEnumerationAccess does not provide single elements like XNameAccess and XIndexAccess, but it creates an enumeration of objects which has methods to go to the next element as long as there are more elements.

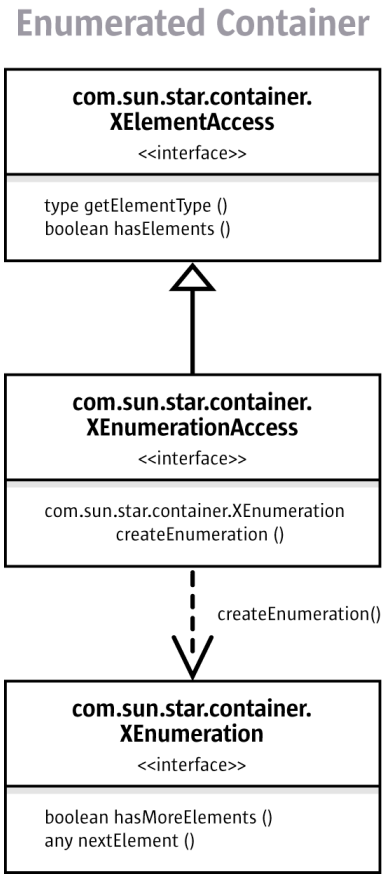


Illustration 7: Enumerated Container

Sets of objects sometimes support all element access methods, some also support only name, index, or enumeration access. Always look up the various types in the API reference to see which access methods are available.

For instance, the method `getSheets()` at the interface `com.sun.star.sheet.XSpreadsheetDocument` is specified to return a `com.sun.star.sheet.XSpreadsheets` interface inherited from `XNameContainer`. In addition, the API reference tells you that the provided object supports a `com.sun.star.sheet.Spreadsheets` service, which defines additional element access interfaces besides `XSpreadsheets`.

Examples that show how to work with `XNameAccess`, `XIndexAccess`, and `XEnumerationAccess` are provided below.

Name Access

The basic interface which hands out elements by name is the `com.sun.star.container.XNameAccess` interface. It has three methods:

```
any getByName( [in] string name)
sequence < string > getElementNames()
boolean hasByName( [in] string name)
```

In the FirstLoadComponent example above, the method `getSheets()` returned a `com.sun.star.sheet.XSpreadsheets` interface, which inherits from `XNameAccess`. Therefore, you could use `getByName()` to obtain the sheet "MySheet" by name from the `XSpreadsheets` container:

```
XSpreadsheets xSpreadsheets = xSpreadsheetDocument.getSheets();

Object sheet = xSpreadsheets.getByName("MySheet");
XSpreadsheet xSpreadsheet = (XSpreadsheet)UnoRuntime.queryInterface(
            XSpreadsheet.class, sheet);

// use XSpreadsheet interface to get the cell A1 at position 0,0 and enter 42 as value
XCell xCell = xSpreadsheet.getCellByPosition(0, 0);
```

Since `getByName()` returns an `any`, you have to use `queryInterface()` before you can call methods at the spreadsheet object.

Index Access

The interface which hands out elements by index is the `com.sun.star.container.XIndexAccess` interface. It has two methods:

```
any getByIndex( [in] long index)
long getCount()
```

The FirstLoadComponent example allows to demonstrate `XIndexAccess`. The API reference tells us that the service returned by `getSheets()` is a `com.sun.star.sheet.Spreadsheet` service and supports not only the interface `com.sun.star.sheet.XSpreadsheets`, but `XIndexAccess` as well. Therefore, the sheets could have been accessed by index and not just by name by performing a query for the `XIndexAccess` interface from our `xSpreadsheets` variable:

```
XIndexAccess xSheetIndexAccess = (XIndexAccess)UnoRuntime.queryInterface(
           XIndexAccess.class, xSpreadsheets);

Object sheet = XSheetIndexAccess.getByIndex(0);
```

Enumeration Access

The interface `com.sun.star.container.XEnumerationAccess` creates enumerations that allow traveling across a set of objects. It has one method:

```
com.sun.star.container.XEnumeration createEnumeration()
```

The enumeration object gained from `createEnumeration()` supports the interface `com.sun.star.container.XEnumeration`. With this interface we can keep pulling elements out of the enumeration as long as it has more elements. `XEnumeration` supplies the methods:

```
boolean hasMoreElements()
any nextElement()
```

which are meant to build loops such as:

```
while (xCells.hasMoreElements()) {

    Object cell = xCells.nextElement();
    // do something with cell
}
```

For example, in spreadsheets you have the opportunity to find out which cells contain formulas. The resulting set of cells is provided as `XEnumerationAccess`.

The interface that queries for cells with formulas is `com.sun.star.sheet.XCellRangesQuery`, it defines (among others) a method

```
XSheetCellRanges queryContentCells(short cellFlags)
```

which queries for cells having content as defined in the constants group com.sun.star.sheet.CellFlags. One of these cell flags is `FORMULA`. From `queryContentCells()` we receive an object with an `com.sun.star.sheet.XSheetCellRanges` interface, which has these methods:

```
    XEnumerationAccess getCells()
    String getRangeAddressesAsString()
    sequence< com.sun.star.table.CellRangeAddress > getRangeAddresses()
```

The method `getCells()` can be used to list all formula cells and the containing formulas in the spreadsheet document from our FirstLoadComponent example, utilizing `XEnumerationAccess`. (FirstSteps/FirstLoadComponent.java)

```
XCellRangesQuery xCellQuery = (XCellRangesQuery)UnoRuntime.queryInterface(
    XCellRangesQuery.class, sheet);
XSheetCellRanges xFormulaCells = xCellQuery.queryContentCells(
    (short)com.sun.star.sheet.CellFlags.FORMULA);

XEnumerationAccess xFormulas = xFormulaCells.getCells();
XEnumeration xFormulaEnum = xFormulas.createEnumeration();

while (xFormulaEnum.hasMoreElements()) {

    Object formulaCell = xFormulaEnum.nextElement();

    // do something with formulaCell
    xCell = (XCell)UnoRuntime.queryInterface(XCell.class, formulaCell);
    XCellAddressable xCellAddress = (XCellAddressable)UnoRuntime.queryInterface(
        XCellAddressable.class, xCell);
    System.out.print("Formula cell in column " + xCellAddress.getCellAddress().Column
        + ", row " + xCellAddress.getCellAddress().Row
        + " contains " + xCell.getFormula());
}
```

2.6 How do I know Which Type I Have?

A common problem is deciding what capabilities an object really has, after you receive it from a method.

By observing the code completion in Java IDE, you can discover the base interface of an object returned from a method. You will notice that `loadComponentFromURL()` returns a `com.sun.star.lang.XComponent`.

By pressing Alt + F1 in the NetBeans IDE you can read specifications about the interfaces and services you are using.

However, methods can only be specified to return one interface type. The interface you get from a method very often supports more interfaces than the one that is returned by the method. Furthermore, the interface does not tell anything about the properties the object contains.

Therefore you should uses this manual to get an idea how things work. Then start writing code, using the code completion and the API reference.

In addition, you can try the InstanceInspector, a Java tool which is part of the OpenOffice.org SDK examples. It is a Java component that can be registered with the office and shows interfaces and properties of the object you are currently working with.

In OpenOffice.org Basic, you can inspect objects using the following Basic properties.

```
sub main
  oDocument = thiscomponent
  msgBox(oDocument.dbg_methods)
  msgBox(oDocument.dbg_properties)
  msgBox(oDocument.dbg_supportedInterfaces)
end sub
```

2.7 Example: Hello Text, Hello Table, Hello Shape

The goal of this section is to give a brief overview of those mechanisms in the OpenOffice.org API, which are common to all document types. The three main application areas of OpenOffice.org are text, tables and drawing shapes. The point is: texts, tables and drawing shapes can occur in all three document types, no matter if you are dealing with a Writer, Calc or Draw/Impress file, but they are treated in the same manner everywhere. When you master the common mechanisms, you will be able to insert and use texts, tables and drawings in all document types.

2.7.1 Common Mechanisms for Text, Tables and Drawings

We want to stress the common ground, therefore we start with the common interfaces and properties that allow to manipulate existing texts, tables and drawings. Afterwards we will demonstrate the different techniques to create text, table and drawings in each document type.

The key interfaces and properties to work with existing texts, tables and drawings are the following:

For *text* the interface `com.sun.star.text.XText` contains the methods that change the actual text and other text contents (examples for text contents besides conventional text paragraphs are text tables, text fields, graphic objects and similar things, but such contents are not available in all contexts). When we talk about text here, we mean any text - text in text documents, text frames, page headers and footers, table cells or in drawing shapes. `XText` is the key for text everywhere in OpenOffice.org.

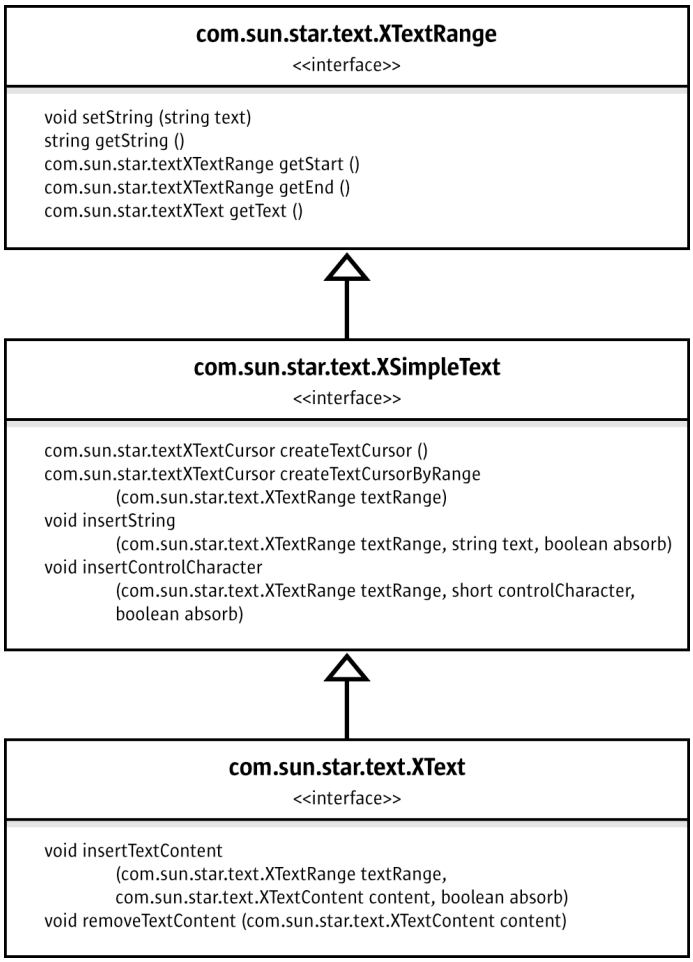


Illustration 8: XTextRange

The interface `com.sun.star.text.XText` has the ability to set or get the text as a single string, and to locate the beginning and the end of a text. Furthermore, `XText` can insert strings at an arbitrary position in the text and create text cursors to select and format text. Finally, `XText` handles text contents through the methods `insertTextContent` and `removeTextContent`, although not all texts accept text contents other than conventional text. In fact, `XText` covers all this by inheriting from `com.sun.star.text.XSimpleText` that is inherited from `com.sun.star.text.XTextRange`.

Text formatting happens through the properties which are described in the services `com.sun.star.style.ParagraphProperties` and `com.sun.star.style.CharacterProperties`.

The following example method `manipulateText()` adds text, then it uses a text cursor to select and format a few words using `CharacterProperties`, afterwards it inserts more text. The method `manipulateText()` only contains the most basic methods of `XText` so that it works with every text object. In particular, it avoids `insertTextContent()`, since there are no text contents except for conventional text that can be inserted in all text objects.(FirstSteps/HelloTextTableShape.java)

```
protected void manipulateText(XText xText) throws com.sun.star.uno.Exception {
        // simply set whole text as one string
        xText.setString("He lay flat on the brown, pine-needled floor of the forest, "
            + "his chin on his folded arms, and high overhead the wind blew in the tops "
            + "of the pine trees.");

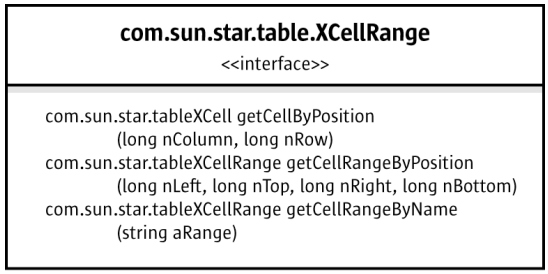
        // create text cursor for selecting and formatting
        XTextCursor xTextCursor = xText.createTextCursor();
        XPropertySet xCursorProps = (XPropertySet)UnoRuntime.queryInterface(
            XPropertySet.class, xTextCursor);

        // use cursor to select "He lay" and apply bold italic
        xTextCursor.gotoStart(false);
```

```
        xTextCursor.goRight((short)6, true);
        // from CharacterProperties
        xCursorProps.setPropertyValue("CharPosture",
            com.sun.star.awt.FontSlant.ITALIC);
        xCursorProps.setPropertyValue("CharWeight",
            new Float(com.sun.star.awt.FontWeight.BOLD));

        // add more text at the end of the text using insertString
        xTextCursor.gotoEnd(false);
        xText.insertString(xTextCursor, " The mountainside sloped gently where he lay; "
            + "but below it was steep and he could see the dark of the oiled road "
            + "winding through the pass. There was a stream alongside the road "
            + "and far down the pass he saw a mill beside the stream and the falling water "
            + "of the dam, white in the summer sunlight.", false);
        // after insertString the cursor is behind the inserted text, insert more text
        xText.insertString(xTextCursor, "\n  \"Is that the mill?\" he asked.", false);
}
```

In *tables and table cells,* the interface `com.sun.star.table.XCellRange` allows you to retrieve single cells and subranges of cells. Once you have a cell, you can work with its formula or numeric value through the interface `com.sun.star.table.XCell`.



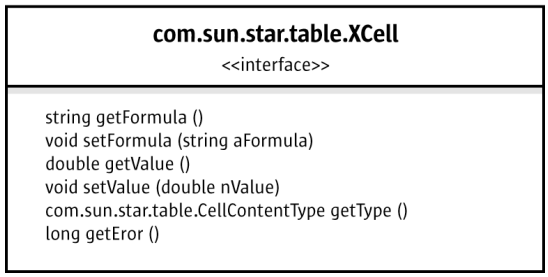


Illustration 9: Cell and Cell Range

Table formatting is partially different in text tables and spreadsheet tables. Text tables use the properties specified in `com.sun.star.text.TextTable`, whereas spreadsheet tables use `com.sun.star.table.CellProperties`. Furthermore there are table cursors that allow to select and format cell ranges and the contained text. But since a `com.sun.star.text.TextTableCursor` works quite differently from a `com.sun.star.sheet.SheetCellCursor`, we will discuss them in the chapters about text and spreadsheet documents.(FirstSteps/HelloTextTableShape.java)

```
protected void manipulateTable(XCellRange xCellRange) throws com.sun.star.uno.Exception {

        String backColorPropertyName = "";
        XPropertySet xTableProps = null;

        // enter column titles and a cell value
        // Enter "Quotation" in A1, "Year" in B1. We use setString because we want to change the whole
        // cell text at once
        XCell xCell = xCellRange.getCellByPosition(0,0);
        XText xCellText = (XText)UnoRuntime.queryInterface(XText.class, xCell);
        xCellText.setString("Quotation");
        xCell = xCellRange.getCellByPosition(1,0);
        xCellText = (XText)UnoRuntime.queryInterface(XText.class, xCell);
        xCellText.setString("Year");

        // cell value
        xCell = xCellRange.getCellByPosition(1,1);
        xCell.setValue(1940);
```

```
        // select the table headers and get the cell properties
        XCellRange xSelectedCells = xCellRange.getCellRangeByName("A1:B1");
        XPropertySet xCellProps = (XPropertySet)UnoRuntime.queryInterface(
            XPropertySet.class, xSelectedCells);

        // format the color of the table headers and table borders
        // we need to distinguish text and spreadsheet tables:
        // - the property name for cell colors is different in text and sheet cells
        // - the common property for table borders is com.sun.star.table.TableBorder, but
        //   we must apply the property TableBorder to the whole text table,
        //   whereas we only want borders for spreadsheet cells with content.

        // XServiceInfo allows to distinguish text tables from spreadsheets
        XServiceInfo xServiceInfo = (XServiceInfo)UnoRuntime.queryInterface(
            XServiceInfo.class, xCellRange);

        // determine the correct property name for background color and the XPropertySet interface
        // for the cells that should get colored border lines
        if (xServiceInfo.supportsService("com.sun.star.sheet.Spreadsheet")) {
            backColorPropertyName = "CellBackColor";
            // select cells
             xSelectedCells = xCellRange.getCellRangeByName("A1:B2");
             // table properties only for selected cells
            xTableProps = (XPropertySet)UnoRuntime.queryInterface(
                XPropertySet.class, xSelectedCells);
        }
        else if (xServiceInfo.supportsService("com.sun.star.text.TextTable")) {
            backColorPropertyName = "BackColor";
           // table properties for whole table
            xTableProps = (XPropertySet)UnoRuntime.queryInterface(
                XPropertySet.class, xCellRange);
        }
        // set cell background color
        xCellProps.setPropertyValue(backColorPropertyName, new Integer(0x99CCFF));

        // set table borders
        // create description for blue line, width 10
        // colors are given in ARGB, comprised of four bytes for alpha-red-green-blue as in 0xAARRGGBB
         BorderLine theLine = new BorderLine();
        theLine.Color = 0x000099;
        theLine.OuterLineWidth = 10;
        // apply line description to all border lines and make them valid
        TableBorder bord = new TableBorder();
        bord.VerticalLine = bord.HorizontalLine =
            bord.LeftLine = bord.RightLine =
            bord.TopLine = bord.BottomLine =
                theLine;
        bord.IsVerticalLineValid = bord.IsHorizontalLineValid =
            bord.IsLeftLineValid = bord.IsRightLineValid =
            bord.IsTopLineValid = bord.IsBottomLineValid =
                true;

        xTableProps.setPropertyValue("TableBorder", bord);

}
```

On *drawing shapes,* the interface `com.sun.star.drawing.XShape` is used to determine the position and size of a shape.

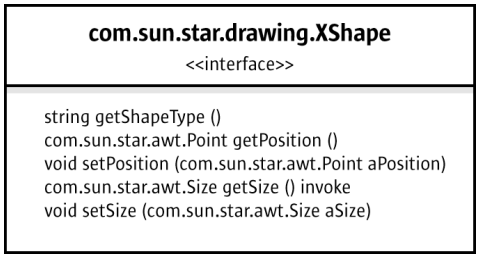


Illustration 10: XShape

Everything else is a matter of property-based formatting and there is a multitude of properties to use. OpenOffice.org comes with eleven different shapes that are the basis for the drawing tools in the GUI (graphical user interface). Six of the shapes have individual properties that reflect their characteristics. The six shapes are:

- `com.sun.star.drawing.EllipseShape` for circles and ellipses.
- `com.sun.star.drawing.RectangleShape` for boxes.

- `com.sun.star.drawing.TextShape` for text boxes.
- `com.sun.star.drawing.CaptionShape` for labeling.
- `com.sun.star.drawing.MeasureShape` for metering.
- `com.sun.star.drawing.ConnectorShape` for lines that can be "glued" to other shapes to draw connecting lines between them.

Five shapes have no individual properties, rather they share the properties defined in the service `com.sun.star.drawing.PolyPolygonBezierDescriptor`:

- `com.sun.star.drawing.LineShape` is for lines and arrows.
- `com.sun.star.drawing.PolyLineShape` is for open shapes formed by straight lines.
- `com.sun.star.drawing.PolyPolygonShape` is for shapes formed by one or more polygons.
- `com.sun.star.drawing.ClosedBezierShape` is for closed bezier shapes.
- `com.sun.star.drawing.PolyPolygonBezierShape` is for combinations of multiple polygon and Bezier shapes.

All of these eleven shapes use the properties from the following services:

- `com.sun.star.drawing.Shape` describes basic properties of all shapes such as the layer a shape belongs to, protection from moving and sizing, style name, 3D transformation and name.
- `com.sun.star.drawing.LineProperties` determines how the lines of a shape look
- `com.sun.star.drawing.Text` has no properties of its own, but includes:
  - `com.sun.star.drawing.TextProperties` that affects numbering, shape growth and text alignment in the cell, text animation and writing direction.
  - `com.sun.star.style.ParagraphProperties` is concerned with paragraph formatting.
  - `com.sun.star.style.CharacterProperties` formats characters
- `com.sun.star.drawing.ShadowProperties` deals with the shadow of a shape.
- `com.sun.star.drawing.RotationDescriptor` sets rotation and shearing of a shape.
- `com.sun.star.drawing.FillProperties` is only for closed shapes and describes how the shape is filled.
- `com.sun.star.presentation.Shape` adds presentation effects to shapes in presentation documents.

Consider the following example showing how these properties work:
(FirstSteps/HelloTextTableShape.java)

```
protected void manipulateShape(XShape xShape) throws com.sun.star.uno.Exception {
        // for usage of setSize and setPosition in interface XShape see method useDraw() below
        XPropertySet xShapeProps = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, xShape);
        // colors are given in ARGB, comprised of four bytes for alpha-red-green-blue as in 0xAARRGGBB
        xShapeProps.setPropertyValue("FillColor", new Integer(0x99CCFF));
        xShapeProps.setPropertyValue("LineColor", new Integer(0x000099));
        // angles are given in hundredth degrees, rotate by 30 degrees
        xShapeProps.setPropertyValue("RotateAngle", new Integer(3000));
}
```

2.7.2 Creating Text, Tables and Drawing Shapes

The three `manipulateXXX` methods above took text, table and shape objects as parameters and altered them. The following methods show how to create such objects in the various document types. Note that all documents have their own service factory to create objects to be inserted into the document. Aside from that it depends very much on the document type how you proceed. This section only demonstrates the different procedures, the explanation can be found in the chapters about Text, Spreadsheet and Drawing Documents.

First, a small convenience method is used to create new documents.
(FirstSteps/HelloTextTableShape.java)

```
protected XComponent newDocComponent(String docType) throws java.lang.Exception {
        String loadUrl = "private:factory/" + docType;
        xRemoteServiceManager = this.getRemoteServiceManager(unoUrl);
        Object desktop = xRemoteServiceManager.createInstanceWithContext(
            "com.sun.star.frame.Desktop", xRemoteContext);
        XComponentLoader xComponentLoader = (XComponentLoader)UnoRuntime.queryInterface(
            XComponentLoader.class, desktop);
        PropertyValue[] loadProps = new PropertyValue[0];
        return xComponentLoader.loadComponentFromURL(loadUrl, "_blank", 0, loadProps);
}
```

Text, Tables and Drawings in Writer

The method `useWriter` creates a writer document and manipulates its text, then uses the document's internal service manager to instantiate a text table and a shape, inserts them and manipulates the table and shape (FirstSteps/HelloTextTableShape.java). Refer to *7 Text Documents* for more detailed information.

```
protected void useWriter() throws java.lang.Exception {
        try {
            // create new writer document and get text, then manipulate text
            XComponent xWriterComponent = newDocComponent("swriter");
            XTextDocument xTextDocument = (XTextDocument)UnoRuntime.queryInterface(
                XTextDocument.class, xWriterComponent);
            XText xText = xTextDocument.getText();

            manipulateText(xText);

            // get internal service factory of the document
            XMultiServiceFactory xWriterFactory = (XMultiServiceFactory)UnoRuntime.queryInterface(
                XMultiServiceFactory.class, xWriterComponent);

            // insert TextTable and get cell text, then manipulate text in cell
            Object table = xWriterFactory.createInstance("com.sun.star.text.TextTable");
            XTextContent xTextContentTable = (XTextContent)UnoRuntime.queryInterface(
                XTextContent.class, table);

            xText.insertTextContent(xText.getEnd(), xTextContentTable, false);

            XCellRange xCellRange = (XCellRange)UnoRuntime.queryInterface(
                XCellRange.class, table);
            XCell xCell = xCellRange.getCellByPosition(0, 1);
            XText xCellText = (XText)UnoRuntime.queryInterface(XText.class, xCell);

            manipulateText(xCellText);
            manipulateTable(xCellRange);

            // insert RectangleShape and get shape text, then manipulate text
            Object writerShape = xWriterFactory.createInstance(
                "com.sun.star.drawing.RectangleShape");
            XShape xWriterShape = (XShape)UnoRuntime.queryInterface(
                XShape.class, writerShape);
            xWriterShape.setSize(new Size(10000, 10000));
            XTextContent xTextContentShape = (XTextContent)UnoRuntime.queryInterface(
                XTextContent.class, writerShape);

            xText.insertTextContent(xText.getEnd(), xTextContentShape, false);

            XPropertySet xShapeProps = (XPropertySet)UnoRuntime.queryInterface(
                XPropertySet.class, writerShape);
            // wrap text inside shape
            xShapeProps.setPropertyValue("TextContourFrame", new Boolean(true));
```

```
            XText xShapeText = (XText)UnoRuntime.queryInterface(XText.class, writerShape);

            manipulateText(xShapeText);
            manipulateShape(xWriterShape);
        }
        catch( com.sun.star.lang.DisposedException e ) { //works from Patch 1
            xRemoteContext = null;
            throw e;
        }

}
```

Text, Tables and Drawings in Calc

The method useCalc creates a calc document, uses its document factory to create a shape and manipulates the cell text, table and shape. The chapter *8 Spreadsheet Documents* treats all aspects of spreadsheets. (FirstSteps/HelloTextTableShape.java)

```
protected void useCalc() throws java.lang.Exception {
        try {
            // create new calc document and manipulate cell text
            XComponent xCalcComponent = newDocComponent("scalc");
            XSpreadsheetDocument  xSpreadsheetDocument  =
                (XSpreadsheetDocument)UnoRuntime.queryInterface(
                    XSpreadsheetDocument .class, xCalcComponent);
            Object sheets = xSpreadsheetDocument.getSheets();
            XIndexAccess xIndexedSheets = (XIndexAccess)UnoRuntime.queryInterface(
                XIndexAccess.class, sheets);
            Object sheet =  xIndexedSheets.getByIndex(0);

            //get cell A2 in first sheet
            XCellRange xSpreadsheetCells = (XCellRange)UnoRuntime.queryInterface(
                XCellRange.class, sheet);
            XCell xCell = xSpreadsheetCells.getCellByPosition(0,1);
            XPropertySet xCellProps = (XPropertySet)UnoRuntime.queryInterface(
                XPropertySet.class, xCell);
            xCellProps.setPropertyValue("IsTextWrapped", new Boolean(true));

            XText xCellText = (XText)UnoRuntime.queryInterface(XText.class, xCell);

            manipulateText(xCellText);
            manipulateTable(xSpreadsheetCells);

            // get internal service factory of the document
            XMultiServiceFactory xCalcFactory = (XMultiServiceFactory)UnoRuntime.queryInterface(
                XMultiServiceFactory.class, xCalcComponent);
            // get Drawpage
            XDrawPageSupplier xDrawPageSupplier =
                 (XDrawPageSupplier)UnoRuntime.queryInterface(XDrawPageSupplier.class, sheet);
            XDrawPage xDrawPage = xDrawPageSupplier.getDrawPage();

            // create and insert RectangleShape and get shape text, then manipulate text
            Object calcShape = xCalcFactory.createInstance(
                "com.sun.star.drawing.RectangleShape");
            XShape xCalcShape = (XShape)UnoRuntime.queryInterface(
                XShape.class, calcShape);
            xCalcShape.setSize(new Size(10000, 10000));
            xCalcShape.setPosition(new Point(7000, 3000));

            xDrawPage.add(xCalcShape);

            XPropertySet xShapeProps = (XPropertySet)UnoRuntime.queryInterface(
                XPropertySet.class, calcShape);
            // wrap text inside shape
            xShapeProps.setPropertyValue("TextContourFrame", new Boolean(true));


            XText xShapeText = (XText)UnoRuntime.queryInterface(XText.class, calcShape);

            manipulateText(xShapeText);
            manipulateShape(xCalcShape);

        }
        catch( com.sun.star.lang.DisposedException e ) { //works from Patch 1
            xRemoteContext = null;
            throw e;
        }

}
```

Drawings and Text in Draw

The method useDraw creates a draw document and uses its document factory to instantiate and add a shape, then it manipulates the shape. The chapter *9 Drawing* casts more light on drawings and presentations. (FirstSteps/HelloTextTableShape.java)

```
protected void useDraw() throws java.lang.Exception {
        try {
            //create new draw document and insert ractangle shape
            XComponent xDrawComponent = newDocComponent("sdraw");
            XDrawPagesSupplier xDrawPagesSupplier =
                (XDrawPagesSupplier)UnoRuntime.queryInterface(
                    XDrawPagesSupplier.class, xDrawComponent);

            Object drawPages = xDrawPagesSupplier.getDrawPages();
            XIndexAccess xIndexedDrawPages = (XIndexAccess)UnoRuntime.queryInterface(
                XIndexAccess.class, drawPages);
            Object drawPage = xIndexedDrawPages.getByIndex(0);
            XDrawPage xDrawPage = (XDrawPage)UnoRuntime.queryInterface(XDrawPage.class, drawPage);

            // get internal service factory of the document
            XMultiServiceFactory xDrawFactory =
                (XMultiServiceFactory)UnoRuntime.queryInterface(
                    XMultiServiceFactory.class, xDrawComponent);

            Object drawShape = xDrawFactory.createInstance(
                "com.sun.star.drawing.RectangleShape");
            XShape xDrawShape = (XShape)UnoRuntime.queryInterface(XShape.class, drawShape);
            xDrawShape.setSize(new Size(10000, 20000));
            xDrawShape.setPosition(new Point(5000, 5000));
            xDrawPage.add(xDrawShape);

            XText xShapeText = (XText)UnoRuntime.queryInterface(XText.class, drawShape);
            XPropertySet xShapeProps = (XPropertySet)UnoRuntime.queryInterface(
                XPropertySet.class, drawShape);

            // wrap text inside shape
            xShapeProps.setPropertyValue("TextContourFrame", new Boolean(true));

            manipulateText(xShapeText);
            manipulateShape(xDrawShape);
        }
        catch( com.sun.star.lang.DisposedException e ) { //works from Patch 1
            xRemoteContext = null;
            throw e;
        }


}
```

3 Professional UNO

This chapter provides in-depth information about UNO and the use of UNO in various programming languages. There are four sections:

- The *3.1 Professional UNO - Introduction* gives an outline of the UNO architecture.
- The section *3.2 Professional UNO - API Concepts* supplies background information on the API reference.
- The section *3.3 Professional UNO - UNO Concepts* describes the mechanics of UNO, i.e. it shows how UNO objects connect and communicate with each other.
- The section *3.4 Professional UNO - UNO Language Bindings* elaborates on the use of UNO from Java, C++, OpenOffice.org Basic and COM automation.

3.1 Introduction

The goal of UNO (Universal Network Objects) is to provide an environment for network objects across programming language and platform boundaries. UNO objects run and communicate everywhere. UNO reaches this goal by providing the following fundamental framework:

- UNO objects are specified in an abstract meta language, called *UNO IDL* (UNO Interface Definition Language), which is similar to CORBA IDL or MIDL. From UNO IDL specifications, language dependent header files and libraries can be generated to implement UNO objects in the target language. UNO objects in the form of compiled and bound libraries are called *components*. Components must support certain base interfaces to be able to run in the UNO environment.
- To instantiate components in a target environment UNO uses a factory concept. This factory is called the s*ervice manager*. It maintains a database of registered components which are known by their name and can be created by name. The service manager might ask Linux to load and instantiate a shared object written in C++ or it might call upon the local Java VM to instantiate a Java class. This is transparent for the developer, there is no need to care about a component's implementation language. Communication takes place exclusively over interface calls as specified in UNO IDL.
- UNO provides *bridges* to send method calls and receive return values between processes and between objects written in different implementation languages. The bridges use a special UNO remote protocol (urp) for this purpose which is supported for sockets and pipes. Both ends of the bridge must be UNO environments, therefore a language-specific UNO runtime environment to connect to another UNO process in any of the supported languages is required. These runtime environments are provided as language bindings.

- Most objects of OpenOffice.org are able to communicate in a UNO environment. The specification for the programmable features of OpenOffice.org is called the OpenOffice.org API.

3.2 API Concepts

The OpenOffice.org API is a language independent approach to specify the functionality of OpenOffice.org. Its main goals are to provide an API to access the functionality of OpenOffice.org, to enable users to extend the functionality by their own solutions and new features, and to make the internal implementation of OpenOffice.org exchangeable.

A long term target on the OpenOffice.org roadmap is to split the existing OpenOffice.org into small components which are combined to provide the complete OpenOffice.org functionality. Such components are manageable, they interact with each other to provide high level features and they are exchangeable with other implementations providing the same functionality, even if these new implementations are implemented in a different programming language. When this target will be reached, the API, the components and the fundamental concepts will provide a construction kit, which makes OpenOffice.org adaptable to a wide range of specialized solutions and not only an office suite with a predefined and static functionality.

This section provides you with a thorough understanding of the concepts behind the OpenOffice.org API. In the API reference there are UNO IDL data types which are unknown outside of the API. The reference provides abstract specifications which sometimes can make you wonder how they map to implementations you can actually use.

The data types of the API reference are explained in *3.2.1 Professional UNO - API Concepts - Data Types*. The relationship between API specifications and OpenOffice.org implementations is treated in *3.2.2 Professional UNO - API Concepts - Understanding the API Reference*.

3.2.1 Data Types

The data types in the API reference are UNO IDL types which have to be mapped to the types of any programming language that can be used with the OpenOffice.org API. In the chapter *2 First Steps* the most important UNO types were introduced. However, there is much more to be said about simple types, interfaces, properties and services in UNO. There are special flags, conditions and relationships between these types which you will want to know if you are working with UNO on a professional level.

This section explains the types of the API reference from the perspective of a developer who wants to use the OpenOffice.org API. If you are interested in writing your own components, and you must define new interfaces and types, please refer to the chapter 4 Writing UNO Components, which describes how to write your own UNO IDL specifications and how to create UNO components.

Simple Types

UNO IDL provides a set of predefined and fundamental base types which are listed in the following table:

UNO IDL Type	Description
`boolean`	`Can be true` or `false`.

UNO IDL Type	Description
`byte`	One-byte type representing a type that is not modified by the UNO runtime during transport (marshaling) over a UNO bridge.
`char`	Represents a unicode character. When this type is mapped to a programming language, the representation depends on the respective hardware or software architecture.
`double`	Processor dependent double.
`float`	Processor dependent float.
`hyper`	64-bit integer type.
`long`	32-bit integer type.
`short`	16-bit integer type.
`string`	Unicode string type.
`type`	Meta type that describes any other UNO IDL types.
`void`	Empty return value, only possible as return value.
`unsigned hyper`	Deprecated. Unsigned 64-bit integer value.
`unsigned long`	Deprecated. Unsigned 32-bit integer value.
`unsigned short`	Deprecated. Unsigned 16-bit integer value.

The chapters about language bindings *3.4.1 Professional UNO - UNO Language Bindings - Java Language Binding*, *3.4.2 Professional UNO - UNO Language Bindings - UNO C++ Binding*, *3.4.3 Professional UNO - UNO Language Bindings - OpenOffice.org Basic* and *3.4.4 Professional UNO - UNO Language Bindings - Automation Bridge* describe how these types are mapped to the types of your target language.

The Any Type

The special type `any` can represent all other known and defined UNO IDL types. In the target languages, the `any` type requires special treatment. There is an `AnyConverter` in Java and special operators in C++. For details, see the section *3.4 Professional UNO - UNO Language Bindings* about language bindings.

Interfaces

Communication between UNO objects is based on object interfaces. Interfaces can be seen from the outside or the inside of an object.

From the *outside* of an object, an interface provides a functionality or special aspect of the object. Interfaces provide access to objects by publishing a set of operations that cover a certain aspect of an object *without telling anything about its internals*.

The concept of interfaces is quite natural and frequently used in everyday life. Interfaces allow the creation of things that fit in with each other without knowing internal details about them. A power plug that fits into a standard socket or a one-size-fits-all working glove are simple examples. They all work by standardizing the minimal conditions that must be met to make things work together.

A more advanced example would be the "remote control aspect" of a simple TV system. One possible feature of a TV system is a remote control. The remote control functions can be described by an `XPower` and an `XChannel` interface. The illustration below shows a `RemoteControl` object with these interfaces:

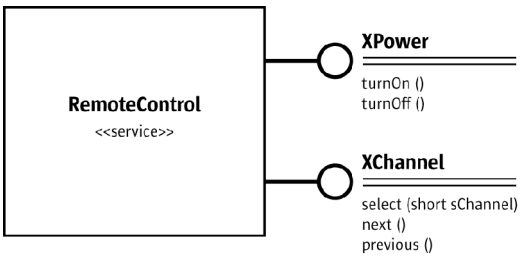


Illustration 11: RemoteControl service

The `XPower` interface has the functions `turnOn()` and `turnOff()` to control the power and the `XChannel` interface has the functions `select()`, `next()`, `previous()` to control the current channel. The user of these interfaces does not care if he uses an original remote control that came with a TV set or a universal remote control as long as it carries out these functions. The user is only dissatisfied if some of the functions promised by the interface do not work with a remote control.

From the *inside* of an object, or from the perspective of someone who implements a UNO object, interfaces are abstract specifications. The abstract specification of all the interfaces in the OpenOffice.org API has the advantage that user and implementer can enter into a contract, agreeing to adhere to the interface specification. A program that strictly uses the OpenOffice.org API according to the specification will always work, while an implementer can do whatever he wants with his objects, as long as he serves the contract.

UNO uses the `interface` type to describe such aspects of UNO objects. All interface names start with the letter X to distinguish them from other types. All interface types must inherit the `com.sun.star.uno.XInterface` interface for basic object communication, either directly or in the inheritance hierarchy. `XInterface` is explained in *3.3.3 Professional UNO - UNO Concepts - Using UNO Interfaces*. The `interface` types define *operations* to provide access to the specified UNO objects.

Interface operations allow access to the data inside an object through dedicated methods (member functions) which encapsulate the data of the object. Interfaces only consist of operations. The operations always have a parameter list and a return value, and they may define exceptions for smart error handling.

The exception concept in the OpenOffice.org API is comparable with the exception concepts known from Java or C++. All operations can raise `com.sun.star.uno.RuntimeExceptions` without explicit specification, but all other exceptions must be specified. UNO exceptions are explained in the section *3.3.6 Professional UNO - UNO Concepts - Exception Handling* below.

Consider the following two examples for interface definitions in UNO IDL notation. UNO IDL interfaces resemble Java interfaces, and operations look similar to Java method signatures. However, note the flags in square brackets in the following example:

```
// base interface for all UNO interfaces

interface XInterface
{
        any queryInterface( [in] type aType );
        [oneway] void acquire();
        [oneway] void release();

};

// fragment of the Interface com.sun.star.io.XInputStream

interface XInputStream: com::sun::star::uno::XInterface
{
    long readBytes( [out] sequence<byte> aData,
                    [in] long nBytesToRead )
                raises( com::sun::star::io::NotConnectedException,
                        com::sun::star::io::BufferSizeExceededException,
                        com::sun::star::io::IOException);
```

```
    ...
};
```

The `[oneway]` flag indicates that an operation will be executed asynchronously. For instance, the method `acquire()` in the interface `com.sun.star.uno.XInterface` is defined to be `oneway`.

There are also parameter flags. Each parameter definition begins with one of the direction flags `in`, `out`, or `inout` to specify the use of the parameter:

- `in` specifies that the parameter will be used as an input parameter only
- `out` specifies that the parameter will be used as an output parameter only
- `inout` specifies that the parameter will be used as an input and output parameter

These parameter flags do not appear in the API reference. The fact that a parameter is an `[out]` or `[inout]` parameter is explained in the method details.

Interfaces consisting of operations form the basis for service specifications.

Services

We have seen that an interface describes only one aspect of an object. However, it is quite common that objects have more than one aspect. UNO uses *services* to specify complete objects which can have many aspects.

A service comprises a set of interfaces and properties that are needed to support a certain functionality. It can include other services as well. Services are abstract specifications which have to be implemented.

From the perspective of a *user* of a UNO object, the object offers one or sometimes even several services described in the API reference. The services are utilized through method calls grouped in interfaces, and through properties, which are handled through special interfaces as well. Because the access to the functionality is provided by interfaces only, the implementation is irrelevant to a user who wants to use a service.

From the perspective of an *implementer* of a UNO object, services are used to define a functionality independently of a programming language and without giving instructions about the internal implementation of the service. Implementing a service means that the component must implement all specified interfaces and properties. It is possible that a UNO object implements more than one service. Sometimes it is useful to implement two or more services because they have related functionality or the services support different views to the component.

Illustration 11 shows the relationship between interfaces, services and components. The language independent specification of a service with several interfaces is used to implement a UNO component that fulfills the specification.

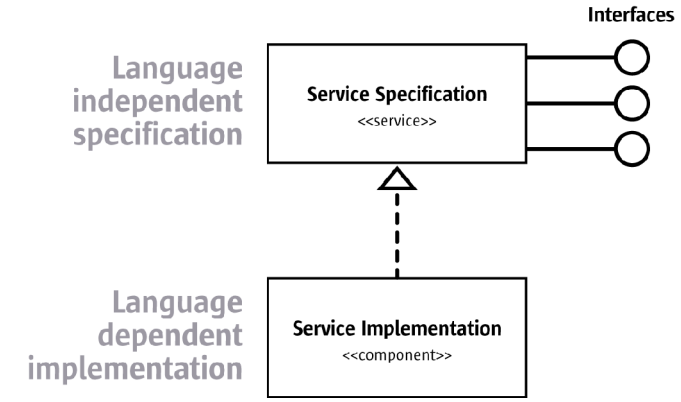


Illustration 12: Interfaces, services and implementation

The functionality of a TV system with a TV set and a remote control can be described in terms of service specifications. The interfaces `XPower` and `XChannel` described above would be part of a service specification `RemoteControl`. The new service `TVSet` consists of the three interfaces `XPower`, `XChannel` and `XStandby` to control the power, the channel selection, the additional power function `standby()` and a `timer()` function.

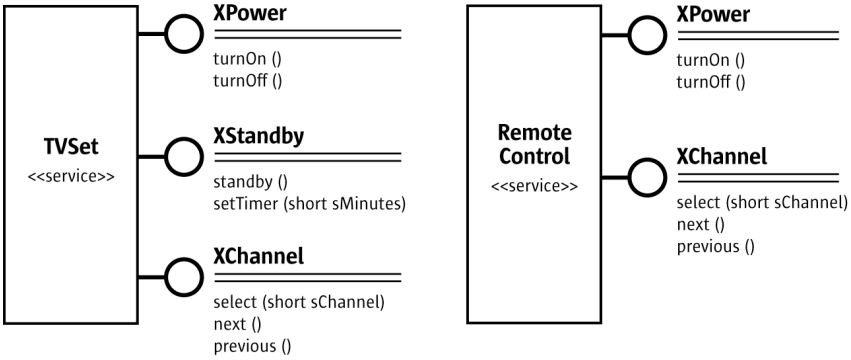


Illustration 13: TV System Specification

Referencing Interfaces

References to interfaces in a service definition mean that an implementation of this service *must* offer the specified interfaces. However, optional interfaces are possible. If a service contains an optional interface, the service may or may not export this interface. If you utilize an optional interface of a UNO object, always check if the result of `queryInterface()` is equal to `null` and react accordingly—otherwise your code will not be compatible with implementations without the optional interface and you might end up with null pointer exceptions. The following UNO IDL snippet shows a fragment of the specification for the `com.sun.star.text.TextDocument` service in the OpenOffice.org API. Note the flag `optional` in square brackets, which makes the interfaces `XFootnotesSupplier` and `XEndnotesSupplier` non-mandatory.

```
// com.sun.star.text.TextDocument
service TextDocument
{
    ...

    interface com::sun::star::text::XTextDocument;
    interface com::sun::star::util::XSearchable;
    interface com::sun::star::util::XRefreshable;
    [optional] interface com::sun::star::text::XFootnotesSupplier;
    [optional] interface com::sun::star::text::XEndnotesSupplier;

    ...
};
```

Including Properties

When the structure of the OpenOffice.org API was founded, the designers discovered that the objects in an office environment would have huge numbers of qualities that did not appear to be part of the structure of the objects, rather they seemed to be superficial changes to the underlying objects. It was also clear that not all qualities would be present in each object of a certain kind. Therefore, instead of defining a complicated pedigree of optional and non-optional interfaces for each and every quality, the concept of properties was introduced. Properties are data in an object that are provided by name over a generic interface for property access, that contains `getPropertyValue()` and `setPropertyValue()` access methods. The concept of properties has other advantages, and there is more to know about properties. Please refer to *3.3.4 Professional UNO - UNO Concepts - Properties* for further information about properties.

Properties are added to a service in its UNO IDL specification. A `property` defines a member variable with a specific type that is accessible at the implementing component by a specific name. It is possible to add further restrictions to a `property` through additional flags. The following service references one interface and three optional properties. All known API types can be valid property types:

```
// com.sun.star.text.TextContent
service TextContent
{
    interface com::sun::star::text::XTextContent;
    [optional, property] com::sun::star::text::TextContentAnchorType AnchorType;
    [optional, readonly, property] sequence<com::sun::star::text::TextContentAnchorType> AnchorTypes;
    [optional, property] com::sun::star::text::WrapTextMode TextWrap;
};
```

Possible property flags are:

- `optional`
  The property does not have to be supported by the implementing component.
- `readonly`
  The value of the property cannot be changed using `com.sun.star.beans.XPropertySet`.
- `bound`
  Changes of property values are broadcast to `com.sun.star.beans.XPropertyChangeListeners`, if any were registered through `com.sun.star.beans.XPropertySet`.
- `constrained`
  The property broadcasts an event before its value changes. Listeners have the right to veto the change.
- `maybeambiguous`
  Possibly the property value cannot be determined in some cases, for example, in multiple selections with different values.
- `maybedefault`
  The value might be stored in a style sheet or in the environment instead of the object itself.

- `maybevoid`
  In addition to the range of the property type, the value can be void. It is similar to a null value in databases.
- `removable`
  The property is removable, this is used for dynamic properties.
- `transient`
  The property will not be stored if the object is serialized

Referencing other Services

Services can include other services. Such references may be optional. That a service is included by another service has nothing to do with implementation inheritance, only the specifications are combined. It is up to the implementer if he inherits or delegates the necessary functionality, or if he implements it from scratch.

The service `com.sun.star.text.Paragraph` in the following UNO IDL example includes one mandatory service `com.sun.star.text.TextContent` and five optional services. Every `Paragraph` must be a `TextContent`. It can be a `TextTable` and it is used to support formatting properties for paragraphs and characters:

```
// com.sun.star.text.Paragraph
service Paragraph
{
    service com::sun::star::text::TextContent;
    [optional] service com::sun::star::text::TextTable;
    [optional] service com::sun::star::style::ParagraphProperties;
    [optional] service com::sun::star::style::CharacterProperties;
    [optional] service com::sun::star::style::CharacterPropertiesAsian;
    [optional] service com::sun::star::style::CharacterPropertiesComplex;

    ...
};
```

Service Implementations in Components

A *component* is a shared library or Java archive containing implementations of one or more services in one of the target programming languages supported by UNO. Such a component must meet basic requirements, mostly different for the different target language, and it must support the specification of the implemented services. That means all specified interfaces and properties must be implemented. Components must be registered in the UNO runtime system. After the registration all implemented services can be used by ordering an instance of the service at the appropriate service factory and accessing the functionality over interfaces.

Based on our example specifications for a `TVSet` and a `RemoteControl` service, a component `RemoteTVImpl` could simulate a remote TV system:

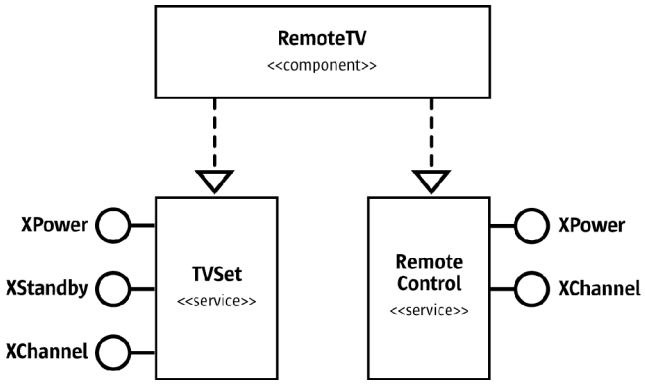


Illustration 14: RemoteTVImpl Component

Such a RemoteTV component could be a jar file or a shared library. It would contain two service implementations, TVSet and RemoteControl. Once the RemoteTV component is registered with the global service manager, users can call the factory method of the service manager and ask for a TVSet or a RemoteControl service. Then they could use their functionality over the interfaces XPower, XChannel and XStandby. When a new implementation of these services with better performance or new features is available later on, the old component can be replaced without breaking existing code, provided that the new features are introduced by adding interfaces.

Structs

A struct type defines several elements in a record. The elements of a struct are UNO IDL types with a unique name within the struct. Structs have the disadvantage not to encapsulate data, but the absence of get () and set() methods can help to avoid the overhead of method calls over a UNO bridge. UNO IDL supports single inheritance for struct types. A derived struct recursively inherits all elements of the parent and its parents.

```
// com.sun.star.lang.EventObject
/** specifies the base for all event objects and identifies the
        source of the event.
 */
struct EventObject
{
        /** refers to the object that fired the event.
         */
        com::sun::star::uno::XInterface Source;

};

// com.sun.star.beans.PropertyChangeEvent
struct PropertyChangeEvent : com::sun::star::lang::EventObject {
    string PropertyName;
    boolean Further;
    long PropertyHandle;
    any OldValue;
    any NewValue;
};
```

Predefined Values

The API offers many predefined values, that are used as method parameters, or returned by methods. In UNO IDL there are two different data types for predefined values: constants and enumerations.

const

A const defines a named value of a valid UNO IDL type. The value depends on the specified type and can be a literal (integer number, floating point number or a character), an identifier of another const type or an arithmetic term using the operators: +, -, *, /, ~, &, |, %, ^, <<, >>.

Since a wide selection of types and values is possible in a const, const is occasionally used to build bit vectors which encode combined values.

```
const short ID = 23;
const boolean ERROR = true;
const double PI = 3.1415;
```

Usually const definitions are part of a constants group.

constants

The `constants` type defines a named group of const values. A const in a constants group is denoted by the group name and the const name. In the UNO IDL example below, `ImageAlign.RIGHT` refers to the value 2:

```
constants ImageAlign {
    const short LEFT = 0;
    const short TOP = 1;
    const short RIGHT = 2;
    const short BOTTOM = 3;
};
```

enum

An `enum` type is equivalent to an enumeration type in C++. It contains an ordered list of one or more identifiers representing long values of the `enum` type. By default, the values are numbered sequentially, beginning with 0 and adding 1 for each new value. If an `enum` value has been assigned a value, all following `enum` values without a predefined value get a value starting from this assigned value.

```
// com.sun.star.uno.TypeClass
enum TypeClass {
    VOID,
    CHAR,
    BOOLEAN,
    BYTE,
    SHORT,
    ...
};

enum Error {
    SYSTEM = 10, // value 10
    RUNTIME,     // value 11
    FATAL,       // value 12
    USER = 30,   // value 30
    SOFT         // value 31
};
```

If enums are used during debugging, you should be able to derive the numeric value of an enum by counting its position in the API reference. However, never use literal numeric values instead of enums in your programs.



Once an enum type has been specified and published, you can trust that it is not extended later on, for that would break existing code. However, new const vaues may be added to a constant group.

Sequences

A `sequence` type is a set of elements of the same type, that has a variable number of elements. In UNO IDL, the used element always references an existing and known type or another `sequence` type. A `sequence` can occur as a normal type in all other type definitions.

```
sequence< com::sun::star::uno::XInterface >
sequence< string > getNamesOfIndex( sequence< long > indexes );
```

Modules

Modules are namespaces, similar to namespaces in C++ or packages in Java. They group services, interfaces, structs, exceptions, enums, typedefs, constant groups and submodules with related functional content or behavior. They are utilized to specify coherent blocks in the API, this allows for a well-structured API. For example, the module `com.sun.star.text` contains interfaces and other types for text handling. Some other typical modules are `com.sun.star.uno`, `com.sun.star.drawing`, `com.sun.star.sheet` and `com.sun.star.table`. Identifiers inside a

module do not clash with identifiers in other modules, therefore it is possible for the same name to occur more than once. The global index of the API reference shows that this does happen.

Although it may seem that the modules correspond with the various parts of OpenOffice.org, there is no direct relationship between the API modules and the OpenOffice.org applications Writer, Calc and Draw. Interfaces from the module `com.sun.star.text` are used in Calc and Draw. Modules like `com.sun.star.style` or `com.sun.star.document` provide generic services and interfaces that are not specific to any one part of OpenOffice.org.

The modules you see in the API reference were defined by nesting UNO IDL types in module instructions. For example, the module `com.sun.star.uno` contains the interface `XInterface`:

```
module com {
    module sun {
        module star {
            module uno {
                interface XInterface {
                    ...
                };
            };
        };
    };
};
```

Exceptions

An `exception` type indicates an error to the caller of a function. The type of an exception gives a basic description of the kind of error that occurred. In addition, the UNO IDL `exception` types contain elements which allow for an exact specification and a detailed description of the error. The `exception` type supports inheritance, this is freqzuently used to define a hierarchy of errors. Exceptions are only used to raise errors, not as method parameters or return types.

UNO IDL requires that all exceptions must inherit from `com.sun.star.uno.Exception`. This is a precondition for the UNO runtime.

```
// com.sun.star.uno.Exception is the base exception for all exceptions
exception Exception {
    string Message;
    Xinterface Context;
};

// com.sun.star.uno.RuntimeException is the base exception for serious problems
// occuring at runtime, usually programming errors or problems in the runtime environment
exception RuntimeException : com::sun::star::uno::Exception {
};

// com.sun.star.uno.SecurityException is a more specific RuntimeException
exception SecurityException : com::sun::star::uno::RuntimeException {
};
```

Exceptions may only be thrown by operations which were specified to do so. In contrast, `com.sun.star.uno.RuntimeExceptions` can always occur.



The methods acquire() and release of the UNO base interface `com.sun.star.uno.XInterface` are an exception to the above rule. They are the only operations that may not even throw runtime exceptions. But in Java and C++ programs, you do not use these methods directly, they are handled by the respective language binding.

Singletons

Singletons are used to specify named objects where exactly *one* instance can exist in the life of a UNO component context. A singleton references one service and specifies that the only existing instance of this service can be reached over the component context using the name of the singleton. If no instance of the service exists, the component context will instantiate a new one.

```
singleton theServiceManager {
    service com::sun::star::lang::ServiceManager
};
```

3.2.2 Understanding the API Reference

Specification, Implementation and Instances

The API specifications you find in the API reference are abstract. The service descriptions of the API reference are not about classes that previously exist somewhere. The specifications are first, then the UNO implementation is created according to the specification. That holds true even for legacy implementations that had to be adapted to UNO.

Moreover, since a component developer is free to implement services and interfaces as required, there is not necessarily a one-to-one relationship between a certain service specification and a real object. The real object can be capable of more things than specified in a service definition. For example, if you order a service at the factory or receive an object from a getter or `getPropertyValue()` method, the specified features will be present, but there may be additional features. For instance, the text document model has a few interfaces which are not included in the specification for the `com.sun.star.text.TextDocument`.

Because of the optional interfaces and properties, it is impossible to comprehend fully from the API reference what a given instance of an object in OpenOffice.org is capable of. The optional interfaces and properties are correct for an abstract specification, but it means that when you leave the scope of mandatory interfaces and properties, the reference only defines how things are allowed to work, not how they actually work.

Another important point is the fact that there are several entry points where service implementations are actually available. You cannot instantiate every service that can be found in the API reference by means of the global service manager. The reasons are:

- Some services need a certain context. For instance, it does not make sense to instantiate a `com.sun.star.text.TextFrame` independently from an existing text document or any other surrounding where it could be of any use. Such services are usually not created by the global service manager, but by document factories which have the necessary knowledge to create objects that work in a certain surrounding. That does not mean you will never be able to get a text frame from the global service manager to insert. So, if you wish to use a service in the API reference, ask yourself where you can get an instance that supports this service, and consider the context in which you want to use it. If the context is a document, it is quite possible that the document factory will be able to create the object.
- Services are not only used to specify possible class implementations. Sometimes they are used to specify nothing but groups of properties that can be referenced by other service implementations. That is, there are services with no interfaces at all. You cannot create such a service at the service manager.
- A few services need special treatment. For example, you cannot ask the service manager to create an instance of a `com.sun.star.text.TextDocument`. You must load it using the method `loadComponentFromUrl()` at the desktop's `com.sun.star.frame.XComponentLoader` interface.

Consequently, it is sometimes confusing to look up a needed functionality in the API reference, for you need a basic understanding how a functionality works, which services are involved, where they are available etc., before you can really utilize the reference. This manual aims at giving you

this understanding about the OpenOffice.org document models, the database integration and the application itself.

Object Composition

Interfaces only support single inheritance and they are all based on `com.sun.star.uno.XInterface`. In the API reference, this is mirrored in the *Base Hierarchy* section of any interface specification. If you look up an interface, always check the base hierarchy section to understand the full range of supported methods. For instance, if you look up `com.sun.star.text.XText`, you see two methods, `insertTextContent()` and `removeTextContent()`, but there are nine more methods provided by the inherited interfaces. The same applies to exceptions and sometimes also to structs, which support single inheritance as well.

The service specifications in the API reference can contain a section *Included Services*, which is similar to the above in that a single included service might encompass a whole world of services. However, the fact that a service is included has nothing to do with class inheritance. In which manner a service implementation technically includes other services, by inheriting from base implementations, by aggregation, some other kind of delegation or simply by reimplementing everything is by no means defined. And it is uninteresting for an API user – he can absolutely rely on the availability of the described functionality, but he must never rely on inner details of the implementation, which classes provide the functionality, where they inherit from and what they delegate to other classes.

3.3 UNO Concepts

Now that you have an advanced understanding of OpenOffice.org API concepts and you understand the specification of UNO objects , we are ready to explore UNO, i.e. to see how UNO objects connect and communicate with each other.

3.3.1 UNO Interprocess Connections

UNO objects in different environments connect via the interprocess bridge. You can execute calls on UNO object instances, that are located in a different process. This is done by converting the method name and the arguments into a byte stream representation, and sending this package to the remote process, for example, through a socket connection. Most of the examples in this manual use the interprocess bridge to communicate with the OpenOffice.org.

This section deals with the creation of UNO interprocess connections using the UNO API.

Starting OpenOffice.org in Listening Mode

Most examples in this developers guide connect to a running OpenOffice.org and perform API calls, which are then executed in OpenOffice.org. By default, the office does not listen on a resource for security reasons. This makes it necessary to make OpenOffice.org listen on an interprocess connection resource, for example, a socket. Currently this can be done in two ways:

- Start the office with an additional parameter:
  `soffice -accept=socket,host=0,port=2002;urp;`
  This string has to be quoted on unix shells, because the semicolon ';' is interpreted by the shells

- Place the same string without '`-accept=`' into a configuration file. You can edit the file *<OfficePath>/share/registry/data/org/openoffice/Setup.xcu*
  and replace the tag
  ```
  <prop oor:name="ooSetupConnectionURL"/>
  ```
  with
  ```
  <prop oor:name="ooSetupConnectionURL">
      <value>socket,host=localhost,port=2002;urp;StarOffice.ServiceManager
      </value>
  </prop>
  ```
  If the tag is not present, add it within the tag
  ```
  <node oor:name="Office"/>
  ```
  This change affects the whole installation. If you want to configure it for a certain user in a network installation, add the same tag within the node `<node oor:name="Office/>` to the file *Setup.xcu* in the user dependent configuration directory *<OfficePath>/user/registry/data/org/openoffice/*

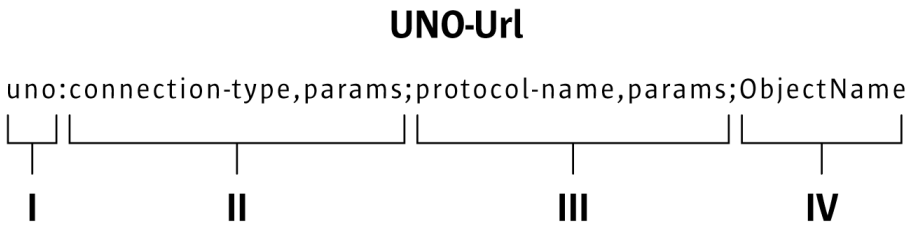
The various parts of the connection URL will be discussed in the next section.

Importing a UNO Object

The most common use case of interprocess connections is to import a reference to a UNO object from an exporting server. For instance, most of the Java examples described in this manual retrieve a reference to the OpenOffice.org `ComponentContext`. The correct way to do this is using the `com.sun.star.bridge.UnoUrlResolver` service. Its main interface `com.sun.star.bridge.XUnoUrlResolver` is defined in the following way:

```
interface XUnoUrlResolver: com::sun::star::uno::XInterface
{
    /** resolves an object on the UNO URL */
    com::sun::star::uno::XInterface resolve( [in] string sUnoUrl )
        raises (com::sun::star::connection::NoConnectException,
                com::sun::star::connection::ConnectionSetupException,
                com::sun::star::lang::IllegalArgumentException);
};
```

The string passed to the `resolve()` method is called a *UNO URL*. It must have the following format:



An example URL could be *uno:socket,host=localhost,port=2002;urp;StarOffice.ServiceManager*. The parts of this URL are:

I. The *URL schema uno:*. This identifies the URL as UNO URL and distinguishes it from others, such as *http:* or *ftp:* URLs.

II. A string which characterizes the *type of connection* to be used to access the other process. Optionally, directly after this string, a comma separated list of name-value pairs can follow, where name and value are separated by a '='. The currently supported connection types are described in *3.3.1 Professional UNO - UNO Concepts - UNO Interprocess Connections - Opening a Connection*. The connection type specifies the transport mechanism used to transfer a byte stream, for example, TCP/IP sockets or named pipes.

III. A string which characterizes the *type of protocol* used to communicate over the established byte stream connection. The string can be followed by a comma separated list of name-value pairs, which can be used to customize the protocol to specific needs. The suggested protocol is urp (UNO Remote Protocol). Some useful parameters are explained below. Refer to the document named *UNO-URL* at *udk.openoffice.org*. for the complete specification.

IV. A process must explicitly export a certain object by a distinct name. It is not possible to access an arbitrary UNO object (which would be possible with IOR in CORBA, for instance).

The following example demonstrates how to import an object using the `UnoUrlResolver`: (ProfUNO/InterprocessConn/UrlResolver.java):

```
XComponentContext xLocalContext =
    com.sun.star.comp.helper.Bootstrap.createInitialComponentContext(null);

// initial serviceManager
XMultiComponentFactory xLocalServiceManager = xLocalContext.getServiceManager();

// create a URL resolver
Object urlResolver = xLocalServiceManager.createInstanceWithContext(
    "com.sun.star.bridge.UnoUrlResolver", xLocalContext);

// query for the XUnoUrlResolver interface
XUnoUrlResolver xUrlResolver =
    (XUnoUrlResolver) UnoRuntime.queryInterface(XUnoUrlResolver.class, urlResolver);

// Import the object
Object rInitialObject = xUrlResolver.resolve(
    "uno:socket,host=localhost,port=2002;urp;StarOffice.ServiceManager");

// XComponentContext
if (null != rInitialObject) {
    System.out.println("initial object successfully retrieved");
} else {
    System.out.println("given initial-object name unknown at server side");
}
```

The usage of the `UnoUrlResolver` has certain disadvantages. You cannot:

- be notified when the bridge terminates for whatever reasons
- close the underlying interprocess connection
- offer a local object as an initial object to the remote process

These issues are addressed by the underlying API, which is explained below. in *3.3.1 Professional UNO - UNO Concepts - UNO Interprocess Connections - Opening a Connection.*

Characteristics of the Interprocess Bridge

The whole bridge is *threadsafe* and allows multiple threads to execute remote calls. The dispatcher thread inside the bridge cannot block because it never executes calls. It instead passes the requests to worker threads.

- A *synchronous* call sends the request through the connection and lets the requesting thread wait for the reply. All calls that have a return value, an `out` parameter, or throw an exceptions other than a `RuntimeException` *must* be synchronous.
- An *asynchronous* (or `oneway`) call sends the request through the connection and immediately returns without waiting for a reply. It is currently specified at the IDL interface if a request is synchronous or asynchronous by using the `[oneway]` modifier.

For synchronous requests, *thread identity* is guaranteed. When process A calls process B, and process B calls process A, the same thread waiting in process A will take over the new request. This avoids deadlocks when the same mutex is locked again. For asynchronous requests, this is not possible because there is no thread waiting in process A. Such requests are executed in a new

thread. The series of calls between two processes is guaranteed. If two asynchronous requests from process A are sent to process B, the second request waits until the first request is finished.

The remote bridge can be started in a mode that disables the oneway feature and thus executes every call as a synchronous call. To do this, the protocol part of the UNO URL on the server and client must be extended by ',`Negotiate=0,forceSynchronous=1`'. For example:

```
soffice -accept=socket,host=0,port=2002;urp,Negotiate=0,forceSynchronous=1;
```

for starting the office and

```
"uno:socket,host=localhost,port=2002;urp,Negotiate=0,forceSynchronous=1;StarOffice.ServiceManager"
```

as UNO URL for connecting to it. This can be useful to avoid deadlocks within OpenOffice.org.



Do not activate this mode unless you experience such problems.

Opening a Connection

The method to import a UNO object using the `UnoUrlResolver` has drawbacks as described in the previous chapter. The layer below the `UnoUrlResolver` offers full flexibility in interprocess connection handling.

UNO interprocess bridges are established on the `com.sun.star.connection.XConnection` interface, which encapsulates a reliable bidirectional byte stream connection (such as a TCP/IP connection).

```
interface XConnection: com::sun::star::uno::XInterface
{
    long read( [out] sequence < byte > aReadBytes , [in] long nBytesToRead )
        raises( com::sun::star::io::IOException );
    void write( [in] sequence < byte > aData )
        raises( com::sun::star::io::IOException );
    void flush( ) raises( com::sun::star::io::IOException );
    void close( ) raises( com::sun::star::io::IOException );
    string getDescription();
};
```

There are different mechanisms to establish an interprocess connection. Most of these mechanisms follow a similar pattern. One process listens on a resource and waits for one or more processes to connect to this resource.

This pattern has been abstracted by the services `com.sun.star.connection.Acceptor` that exports the `com.sun.star.connection.XAcceptor` interface and `com.sun.star.connection.Connector` that exports the `com.sun.star.connection.XConnector` interface.

```
interface XAcceptor: com::sun::star::uno::XInterface
{
    XConnection accept( [in] string sConnectionDescription )
        raises( AlreadyAcceptingException,
                ConnectionSetupException,
                com::sun::star::lang::IllegalArgumentException);

    void stopAccepting();
};

interface XConnector: com::sun::star::uno::XInterface
{
    XConnection connect( [in] string sConnectionDescription )
        raises( NoConnectException,ConnectionSetupException );
};
```

The acceptor service is used in the listening process while the connector service is used in the actively connecting service. The methods `accept()` and `connect()` get the connection string as a parameter. This is the connection part of the UNO URL (between *uno:* and *;urp*).

The connection string consists of a connection type followed by a comma separated list of name-value pairs. The following table shows the connection types that are supported by default.

Connection type		
socket	Reliable TCP/IP socket connection	
	Parameter	**Description**
	host	Hostname or IP number of the resource to listen on/connect. May be localhost. In an acceptor string, this may be 0 ('host=0'), which means, that it accepts on all available network interfaces.
	port	TCP/IP port number to listen on/connect to.
	tcpNoDelay	Corresponds to the socket option tcpNoDelay. For a UNO connection, this parameter should be set to 1 (this is NOT the default —it must be added explicitly). If the default is used (0), it may come to 200 ms delays at certain call combinations.
pipe	A named pipe (uses shared memory). This type of interprocess connection is marginally faster than socket connections and works only if both processes are located on the same machine. It does not work on Java by default, because Java does not support named pipes directly	
	Parameter	**Description**
	name	Name of the named pipe. Can only accept one process on name on one machine at a time.



You can add more kinds of interprocess connections by implementing connector and acceptor services, and choosing the service name by the scheme `com.sun.star.connection.Connector.<connection-type>`, where `<connection-type>` is the name of the new connection type.

If you implemented the service `com.sun.star.connection.Connector.mytype`, use the `UnoUrlResolver` with the URL 'uno:mytype,param1=foo;urp;StarOffice.ServiceManager' to establish the interprocess connection to the office.

Creating the Bridge

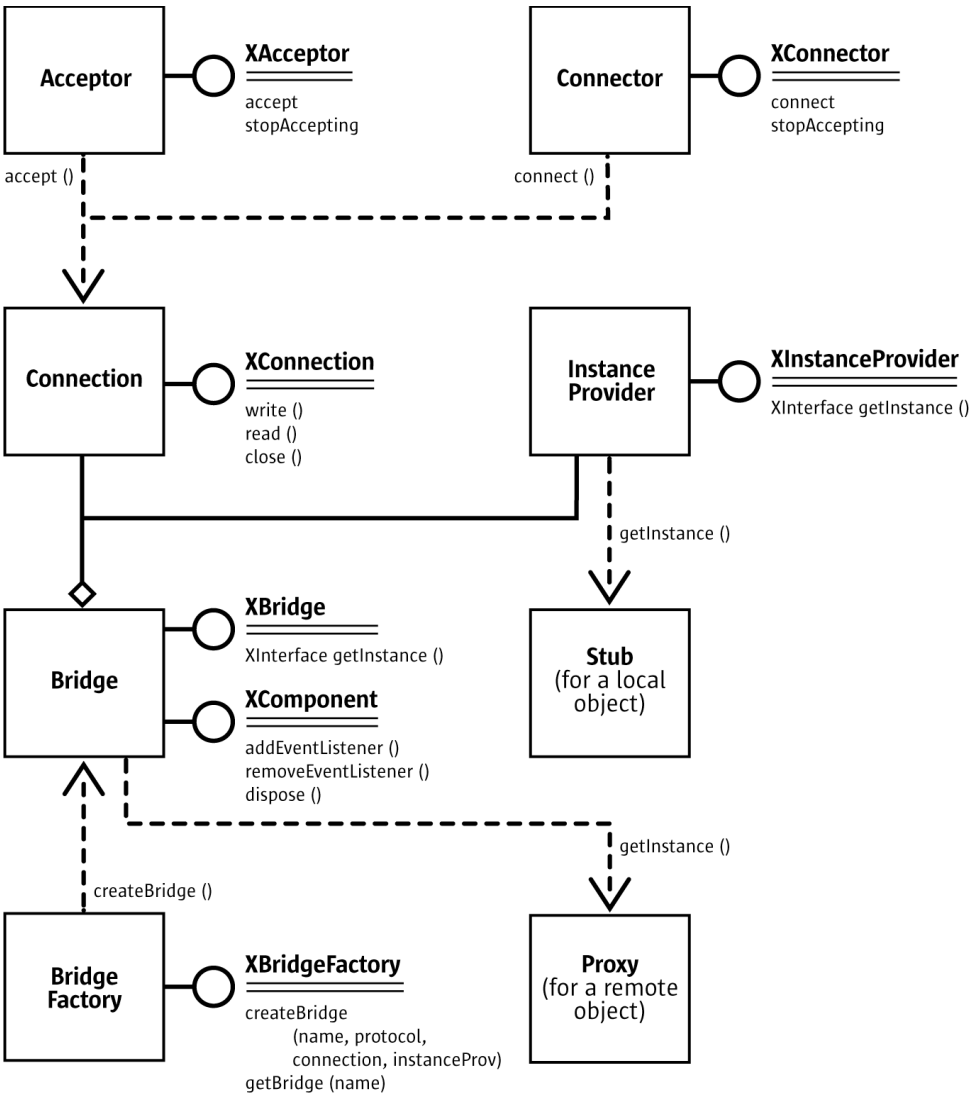


Illustration 15: The interaction of services that are needed to initiate a UNO interprocess bridge. The interfaces have been simplified.

The `XConnection` instance can now be used to establish a UNO interprocess bridge on top of the connection, regardless if the connection was established with a `Connector` or `Acceptor` service (or another method). To do this, you must instantiate the service `com.sun.star.bridge.BridgeFactory`. It supports the `com.sun.star.bridge.XBridgeFactory` interface.

```
interface XBridgeFactory: com::sun::star::uno::XInterface
{
    XBridge createBridge(
            [in] string sName,
            [in] string sProtocol ,
            [in] com::sun::star::connection::XConnection aConnection ,
            [in] XInstanceProvider anInstanceProvider )
        raises ( BridgeExistsException , com::sun::star::lang::IllegalArgumentException );
    XBridge getBridge( [in] string  sName );
    sequence < XBridge > getExistingBridges( );
};
```

The `BridgeFactory` service administrates all UNO interprocess connections. The `createBridge()` method creates a new bridge:

- You can give the bridge a distinct name with the `sName` argument. Later the bridge can be retrieved by using the `getBridge()` method with this name. This allows two independent code pieces to share the same interprocess bridge. If you call `createBridge()` with the name of an already working interprocess bridge, a `BridgeExistsException` is thrown. When you pass an empty string, you always create a new anonymous bridge, which can never be retrieved by `getBridge()` and which never throws a `BridgeExistsException`.
- The second parameter specifies the protocol to be used on the connection. Currently, only the 'urp' protocol is supported. In the UNO URL, this string is separated by two ';'. The urp string may be followed by a comma separated list of name-value pairs describing properties for the bridge protocol. The urp specification can be found on *udk.openoffice.org*.
- The third parameter is the `XConnection` interface as it was retrieved by Connector/Acceptor service.
- The fourth parameter is a UNO object, which supports the `com.sun.star.bridge.XInstanceProvider` interface. This parameter may be a null reference if you do not want to export a local object to the remote process.

```
interface XInstanceProvider: com::sun::star::uno::XInterface
{
    com::sun::star::uno::XInterface getInstance( [in] string sInstanceName )
        raises ( com::sun::star::container::NoSuchElementException );
};
```

The `BridgeFactory` returns a `com.sun.star.bridge.XBridge` interface.

```
interface XBridge: com::sun::star::uno::XInterface
{
    XInterface getInstance( [in] string sInstanceName );
    string getName();
    string getDescription();
};
```

The `XBridge.getInstance()` method retrieves an *initial object* from the remote counterpart. The local `XBridge.getInstance()` call arrives in the remote process as an `XInstanceProvider.getInstance()` call. The object returned can be controlled by the string `sInstanceName`. It completely depends on the implementation of `XInstanceProvider`, which object it returns.

The `XBridge` interface can be queried for a `com.sun.star.lang.XComponent` interface, that adds a `com.sun.star.lang.XEventListener` to the bridge. This listener will be terminated when the underlying connection closes (see above). You can also call `dispose()` on the `XComponent` interface explicitly, which closes the underlying connection and initiates the bridge shutdown procedure.

Closing a Connection

The closure of an interprocess connection can occur for the following reasons:

- The bridge is not used anymore. The interprocess bridge will close the connection when all the proxies to remote objects and all stubs to local objects have been released. This is the normal way for a remote bridge to destroy itself. The user of the interprocess bridge does not need to close the interprocess connection directly—it is done automatically. When one of the communicating processes is implemented in Java, the closure of a bridge is delayed to that point in time when the VM finalizes the last proxies/stubs. Therefore it is unspecified when the interprocess bridge will be closed.
- The interprocess bridge is directly disposed by calling its `dispose()` method.
- The remote counterpart process crashes.

- The connection fails. For example, failure may be due to a dialup internet connection going down.
- An error in marshalling/unmarshalling occurs due to a bug in the interprocess bridge implementation, or an IDL type is not available in one of the processes.

Except for the first reason, all other connection closures initiate an interprocess bridge shutdown procedure. All pending synchronous requests abort with a `com.sun.star.lang.DisposedException`, which is derived from the `com.sun.star.uno.RuntimeException`. Every call that is initiated on a disposed proxy throws a `DisposedException`. After all threads have left the bridge (there may be a synchronous call from the former remote counterpart in the process), the bridge explicitly releases all stubs to the original objects in the local process, which were previously held by the former remote counterpart. The bridge then notifies all registered listeners about the disposed state using `com.sun.star.lang.XEventListener`. The example code for a connection-aware client below shows how to use this mechanism. The bridge itself is destroyed, after the last proxy has been released.

Unfortunately, the various listed error conditions are not distinguishable.

Example: A Connection Aware Client

The following example shows an advanced client which can be informed about the status of the remote bridge. A complete example for a simple client is given in the chapter *2 First Steps*.

The following Java example opens a small awt window containing the buttons **new writer** and **new calc** that opens a new document and a status label. It connects to a running office when a button is clicked for the first time. Therefore it uses the connector/bridge factory combination, and registers itself as an event listener at the interprocess bridge.

When the office is terminated, the disposing event is terminated, and the Java program sets the text in the status label to 'disconnected' and clears the office desktop reference. The next time a button is pressed, the program knows that it has to re-establish the connection.

The method `getComponentLoader()` retrieves the `XComponentLoader` reference on demand:

(ProfUNO/InterprocessConn/ConnectionAwareClient.java)

```
XComponentLoader _officeComponentLoader = null;

// local component context
XComponentContext _ctx;

protected com.sun.star.frame.XComponentLoader getComponentLoader()
        throws com.sun.star.uno.Exception {
    XComponentLoader officeComponentLoader = _officeComponentLoader;

    if (officeComponentLoader == null) {
        // instantiate connector service
        Object x = _ctx.getServiceManager().createInstanceWithContext(
            "com.sun.star.connection.Connector", _ctx);

        XConnector xConnector = (XConnector) UnoRuntime.queryInterface(XConnector.class, x);

        // helper function to parse the UNO URL into a string array
        String a[] = parseUnoUrl(_url);
        if (null == a) {
            throw new com.sun.star.uno.Exception("Couldn't parse UNO URL "+ _url);
        }

        // connect using the connection string part of the UNO URL only.
        XConnection connection = xConnector.connect(a[0]);

        x = _ctx.getServiceManager().createInstanceWithContext(
            "com.sun.star.bridge.BridgeFactory", _ctx);

        XBridgeFactory xBridgeFactory = (XBridgeFactory) UnoRuntime.queryInterface(
            XBridgeFactory.class , x);
```

```
            // create a nameless bridge with no instance provider
            // using the middle part of the UNO URL
            XBridge bridge = xBridgeFactory.createBridge("" , a[1] , connection , null);

            // query for the XComponent interface and add this as event listener
            XComponent xComponent = (XComponent) UnoRuntime.queryInterface(
                XComponent.class, bridge);
            xComponent.addEventListener(this);

            // get the remote instance
            x = bridge.getInstance(a[2]);

            // Did the remote server export this object ?
            if (null == x) {
                throw new com.sun.star.uno.Exception(
                    "Server didn't provide an instance for" + a[2], null);
            }

            // Query the initial object for its main factory interface
            XMultiComponentFactory xOfficeMultiComponentFactory = (XMultiComponentFactory)
                UnoRuntime.queryInterface(XMultiComponentFactory.class, x);

            // retrieve the component context (it's not yet exported from the office)
            // Query for the XPropertySet interface.
            XPropertySet xProperySet = (XPropertySet)
                UnoRuntime.queryInterface(XPropertySet.class, xOfficeMultiComponentFactory);

            // Get the default context from the office server.
            Object oDefaultContext =
                xProperySet.getPropertyValue("DefaultContext");

            // Query for the interface XComponentContext.
            XComponentContext xOfficeComponentContext =
                (XComponentContext) UnoRuntime.queryInterface(
                    XComponentContext.class, oDefaultContext);


            // now create the desktop service
            // NOTE: use the office component context here !
            Object oDesktop = xOfficeMultiComponentFactory.createInstanceWithContext(
                "com.sun.star.frame.Desktop", xOfficeComponentContext);

            officeComponentLoader = (XComponentLoader)
                UnoRuntime.queryInterface( XComponentLoader.class, oDesktop);

            if (officeComponentLoader == null) {
                throw new com.sun.star.uno.Exception(
                    "Couldn't instantiate com.sun.star.frame.Desktop" , null);
            }
            _officeComponentLoader = officeComponentLoader;
        }
        return officeComponentLoader;
    }
```

This is the button event handler:

```
    public void actionPerformed(ActionEvent event) {
        try {
            String sUrl;
            if (event.getSource() == _btnWriter) {
                sUrl = "private:factory/swriter";
            } else {
                sUrl = "private:factory/scalc";
            }
            getComponentLoader().loadComponentFromURL(
                sUrl, "_blank", 0,new com.sun.star.beans.PropertyValue[0]);
            _txtLabel.setText("connected");
        } catch (com.sun.star.connection.NoConnectException exc) {
            _txtLabel.setText(exc.getMessage());
        } catch (com.sun.star.uno.Exception exc) {
            _txtLabel.setText(exc.getMessage());
            exc.printStackTrace();
            throw new java.lang.RuntimeException(exc.getMessage());
        }
    }
```

And the disposing handler clears the `_officeComponentLoader` reference:

```
    public void disposing(com.sun.star.lang.EventObject event) {
        // remote bridge has gone down, because the office crashed or was terminated.
        _officeComponentLoader = null;
        _txtLabel.setText("disconnected");
    }
```

3.3.2 Service Manager and Component Context

This chapter discusses the root object for connections to OpenOffice.org (and to any UNO application) – the service manager. The root object serves as the entry point for every UNO application and is passed to every UNO component during instantiation.

Two different concepts to get the root object currently exist. StarOffice6.0 and OpenOffice.org1.0 use the previous concept. Newer versions or product patches use the the newer concept and provide the previous concept for compatibility issues only. First we will look at the previous concept, the *service manager* as it is used in the main parts of the underlying OpenOffice.org implementation of this guide. Second, we will introduce the *component context*—which is the newer concept and explain the migration path.

Service Manager

The `com.sun.star.lang.ServiceManager` is the main *factory* in every UNO application. It instantiates services by their service name, to enumerate all implementations of a certain service, and to add or remove factories for a certain service at runtime. The service manager is passed to every UNO component during instantiation.

XMultiServiceFactory Interface

The main interface of the service manager is the `com.sun.star.lang.XMultiServiceFactory` interface. It offers three methods: `createInstance()`, `createInstanceWithArguments()` and `getAvailableServiceNames()`.

```
interface XMultiServiceFactory: com::sun::star::uno::XInterface
{
    com::sun::star::uno::XInterface createInstance( [in] string aServiceSpecifier )
        raises( com::sun::star::uno::Exception );

    com::sun::star::uno::XInterface createInstanceWithArguments(
            [in] string ServiceSpecifier,
            [in] sequence<any> Arguments )
        raises( com::sun::star::uno::Exception );

    sequence<string> getAvailableServiceNames();
};
```

- `createInstance()` returns a default constructed service instance. The returned service is guaranteed to support at least all interfaces, which were specified for the requested servicename. The returned `XInterface` reference can now be queried for the interfaces specified at the service description.

  When using the service name, the caller does not have any influence on which concrete implementation is instantiated. If multiple implementations for a service exist, the service manager is free to decide which one to employ. This in general does not make a difference to the caller because every implementation does fulfill the service contract. Performance or other details may make a difference. So it is also possible to pass the *implementation name* instead of the service name, but it is not advised to do so as the implementation name may change.

  In case the service manager does not provide an implementation for a request, a null reference is returned, so it is mandatory to check. Every UNO exception may be thrown during instantiation. Some may be described in the specification of the service that is to be instantiated, for instance, because of a misconfiguration of the concrete implementation. Another reason may be the lack of a certain bridge, for instance the Java-C++ bridge, in case a Java component shall be instantiated from C++ code.

- `createInstanceWithArguments()` instantiates the service with additional parameters. A service signals that it expects parameters during instantiation by supporting the `com.sun.star.lang.XInitialization` interface. The service definition should describe the meaning of each element of the sequence. There maybe services which can only be instantiated with parameters.
- `getAvailableServiceNames()` returns every servicename the service manager does support.

XContentEnumerationAccess Interface

The `com.sun.star.container.XContentEnumerationAccess` interface allows the creation of an enumeration of all implementations of a concrete servicename.

```
interface XContentEnumerationAccess: com::sun::star::uno::XInterface
{
    com::sun::star::container::XEnumeration createContentEnumeration( [in] string aServiceName );

    sequence<string> getAvailableServiceNames();

};
```

The `createContentEnumeration()` method returns a `com.sun.star.container.XEnumeration` interface. Note that it may return an empty reference in case the enumeration is empty.

```
interface XEnumeration: com::sun::star::uno::XInterface
{
    boolean hasMoreElements();

    any nextElement()
        raises( com::sun::star::container::NoSuchElementException,
                com::sun::star::lang::WrappedTargetException );

};
```

In the above case, the returned `any` of the method `Xenumeration.nextElement()` contains a `com.sun.star.lang.XSingleServiceFactory` interface for each implementation of this specific service. You can, for instance, iterate over all implementations of a certain service and check each one for additional implemented services. The `XSingleServiceFactory` interface provides such a method. With this method, you can instantiate a feature rich implementation of a service.

XSet Interface

The `com.sun.star.container.XSet` interface allows the insertion or removal of `com.sun.star.lang.XSingleServiceFactory` or `com.sun.star.lang.XSingleComponentFactory` implementations to the service manager at runtime without making the changes permanent. When the office application terminates, all the changes are lost. The object must also support the `com.sun.star.lang.XServiceInfo` interface that provides information about the implementation name and supported services of the component implementation.

This feature may be of particular interest during the development phase. For instance, you can connect to a running office, insert a new factory into the service manager and directly instantiate the new service without having it registered before.

The chapter *4.9.6 Writing UNO Components - Deployment Options for Components - Special Service Manager Configurations* shows an example that demonstrates how a factory is inserted into the service manager.

Component Context

The service manager was described above as the main factory that is passed to every new instantiated component. Often a component needs more functionality or information that must be

exchangeable after deployment of an application. In this context, the service manager approach is limited.

Therefore, the concept of the *component context* was created. In future, it will be the central object in every UNO application. It is basically a read-only container offering named values. One of the named values is the service manager. The component context is passed to a component during its instantiation. This can be understood as an environment where components live (the relationship is similar to shell environment variables and an executable program).

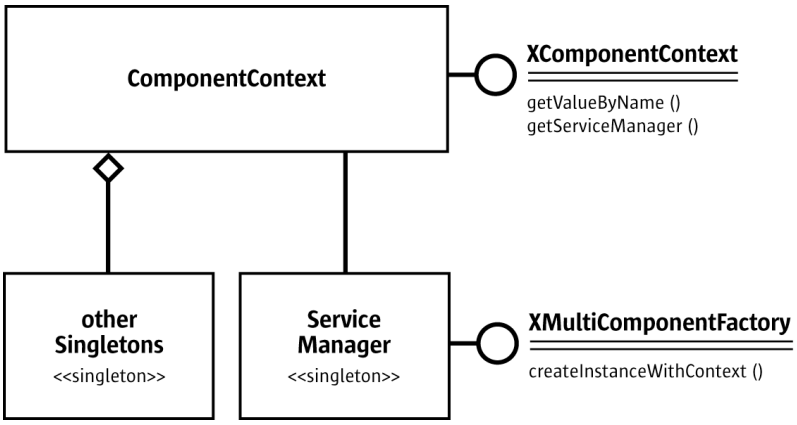


Illustration 16: ComponentContext and the ServiceManager

ComponentContext API

The component context only supports the `com.sun.star.uno.XComponentContext` interface.

```
// module com::sun::star::uno
interface XComponentContext : XInterface
{
    any getValueByName( [in] string Name );
    com::sun::star::lang::XMultiComponentFactory getServiceManager();
};
```

The `getValueByName()` method returns a named value. The `getServiceManager()` is a convenient way to retrieve the value named `/singleton/com.sun.star.lang.theServiceManager`. It returns the `ServiceManager` singleton, because most components need to access the service manager. The component context offers at least three kinds of named values:

Singletons (/singletons/...)
: The singleton concept was introduced in *3.2.1 Professional UNO - API Concepts - Data Types*. In OpenOffice.org 1.0.2 there is only the `ServiceManager` singleton. From OpenOffice.org 1.1, a singleton `/singletons/com.sun.star.util.theMacroExpander` has been added, which can be used to expand macros in configuration files. Other possible singletons can be found in the IDL reference.

Implementation properties (not yet defined)
: These properties customize a certain implementation and are specified in the module description of each component. A module description is an xml-based description of a module (DLL or jar file) which contains the formal description of one or more components.

Service properties (not yet defined)
: These properties can customize a certain service independent from the implementation and are specified in the IDL specification of a service.
Note that service context properties are different from service properties. Service context properties are not subject to change and are the same for every instance of the service that shares the

same component context. Service properties are different for each instance and can be changed at runtime through the `XPropertySet` interface.

Note, that in the scheme above, the `ComponentContext` has a reference to the service manager, but not conversely.

Besides the interfaces discussed above, the `ServiceManager` supports the `com.sun.star.lang.XMultiComponentFactory` interface.

```
interface XMultiComponentFactory : com::sun::star::uno::XInterface
{
        com::sun::star::uno::XInterface createInstanceWithContext(
                [in] string aServiceSpecifier,
                [in] com::sun::star::uno::XComponentContext Context )
                raises (com::sun::star::uno::Exception);

        com::sun::star::uno::XInterface createInstanceWithArgumentsAndContext(
                [in] string ServiceSpecifier,
                [in] sequence<any> Arguments,
                [in] com::sun::star::uno::XComponentContext Context )
                raises (com::sun::star::uno::Exception);

        sequence< string > getAvailableServiceNames();
};
```

It replaces the `XMultiServiceFactory` interface. It has an additional `XComponentContext` parameter for the two object creation methods. This parameter enables the caller to define the component context that the new instance of the component receives. Most components use their initial component context to instantiate new components. This allows for *context propagation.*

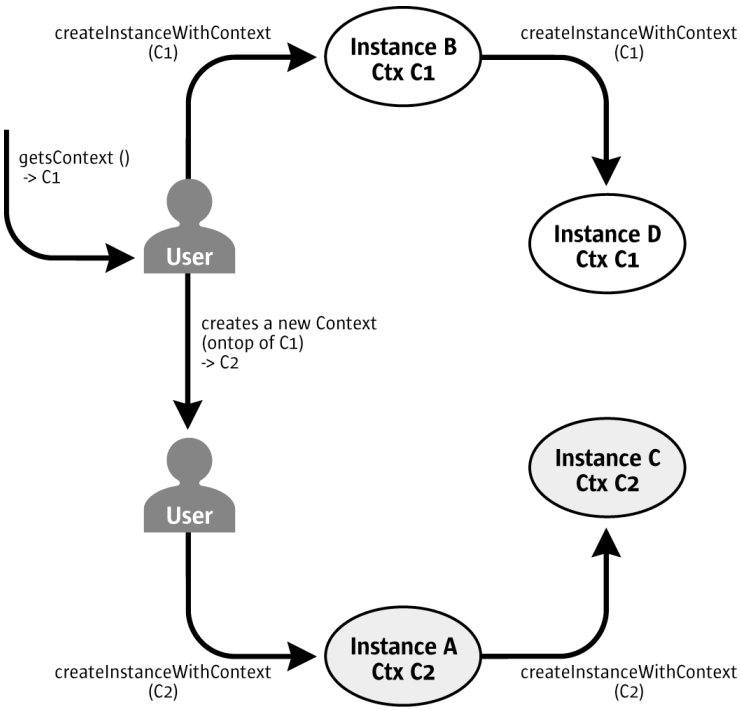


Illustration 17: Context propagation.

The illustration above shows the context propagation. A user might want a special component to get a customized context. Therefore, the user creates a new context by simply wrapping an existing one. The user overrides the desired values and delegates the properties that he is not interested into the original C1 context.The user defines which context Instance A and B receive. Instance A and B propagate their context to every new object that they create. Thus, the user has established two instance trees, the first tree completely uses the context Ctx C1, while the second tree uses Ctx C2.

Availability

The final API for the component context is available in StarOffice 6.0 and OpenOffice 1.0. Use this API instead of the API explained in the service manager section. Currently the component context does not have a persistent storage, so named values can not be added to the context of a deployed OpenOffice.org. Presently, there is no additional benefit from the new API until there is a future release.

Compatibility Issues and Migration Path

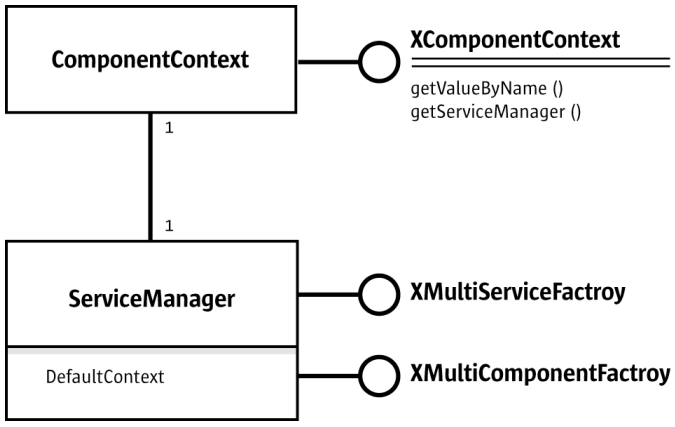


Illustration 18Compromise between service-manger-only und component context concept

As discussed previously, both concepts are currently used within the office. The `ServiceManager` supports the interfaces `com.sun.star.lang.XMultiServiceFactory` and `com.sun.star.lang.XMultiComponentFactory`. Calls to the `XMultiServiceFactory` interface are delegated to the `XMultiComponentFactory` interface. The service manager uses its own `XComponentContext` reference to fill the missing parameter. The component context of the `ServiceManager` can be retrieved through the `XPropertySet` interface as 'DefaultContext'.

```
// Query for the XPropertySet interface.
// Note xOfficeServiceManager is the object retrieved by the
// UNO URL resolver
XPropertySet xPropertySet = (XPropertySet)
        UnoRuntime.queryInterface(XPropertySet.class, xOfficeServiceManager);

// Get the default context from the office server.
Object oDefaultContext = xpropertysetMultiComponentFactory.getPropertyValue("DefaultContext");

// Query for the interface XComponentContext.
xComponentContext = (XComponentContext) UnoRuntime.queryInterface(
        XComponentContext.class, objectDefaultContext);
```

This solution allows the use of the same service manager instance, regardless if it uses the old or new style API. In future, the whole OpenOffice.org code will only use the new API. However, the old API will still remain to ensure compatibility.



The described compromise has a drawback. The service manager now knows the component context, that was not necessary in the original design. Thus, every component that uses the old API (plain `createInstance()`) breaks the context propagation (see Illustration 14). Therefore, it is recommended to use the new API in every new piece of code that is written.

3.3.3 Using UNO Interfaces

Every UNO object must inherit from the interface `com.sun.star.uno.XInterface`. Before using an object, know how to use it and how long it will function. By prescribing `XInterface` to be the base interface for each and every UNO interface, UNO lays the groundwork for object communication.

```
// module com::sun::star::uno
interface XInterface
{
    any queryInterface( [in] type aType );
    [oneway] void acquire();
    [oneway] void release();
};
```

The methods `acquire()` and `release()` handle the lifetime of the UNO object by reference counting. Detailed information about Reference counting is discussed in chapter *3.3.7 Professional UNO - UNO Concepts - Lifetime of UNO Objects*. All current language bindings take care of `acquire()` and `release()` internally whenever there is a reference to a UNO object.

The `queryInterface()` method obtains other interfaces exported by the object. The caller asks the implementation of the object if it supports the interface specified by the type argument. The `type` parameter is an UNO IDL base type, and generally stores the name of a type and its `com.sun.star.uno.TypeClass`. The call may return with an interface reference of the requested type or with a void `any`. In C++ or Java simply test if the result is not equal null.

Unknowingly, we encountered `XInterface` when the service manager was asked to create a service instance:

```
    XComponentContext xLocalContext =
        com.sun.star.comp.helper.Bootstrap.createInitialComponentContext(null);

    // initial serviceManager
    XMultiComponentFactory xLocalServiceManager = xLocalContext.getServiceManager();

    // create a urlresolver
    Object urlResolver  = xLocalServiceManager.createInstanceWithContext(
        "com.sun.star.bridge.UnoUrlResolver", xLocalContext);
```

The IDL specification of `XmultiComponentFactory` shows:

```
// module com::sun::star::lang
interface XMultiComponentFactory : com::sun::star::uno::XInterface
{
    com::sun::star::uno::XInterface createInstanceWithContext(
            [in] string aServiceSpecifier,
            [in] com::sun::star::uno::XComponentContext Context )
        raises (com::sun::star::uno::Exception);
    ...
}
```

The above code shows that `createInstanceWithContext()` provides an instance of the given service, but it only returns a `com.sun.star.uno.XInterface`. This is mapped to java.lang.Object by the Java UNO binding afterwards.

In order to access a service, you need to know which interfaces the service exports. This information is available in the IDL reference. For instance, for the `com.sun.star.bridge.UnoUrlResolver` service, you learn:

```
// module com::sun::star::bridge
service UnoUrlResolver
{
    interface com::sun::star::bridge::XUnoUrlResolver;
};
```

This means the service you ordered at the service manager must support `com.sun.star.bridge.XUnoUrlResolver`. Next *query* the returned object for this interface:

```
// query urlResolver for its com.sun.star.bridge.XUnoUrlResolver interface
XUnoUrlResolver xUrlResolver = (XUnoUrlResolver)
    UnoRuntime.queryInterface(UnoUrlResolver.class, urlResolver);
```

```
// test if the interface was available
if (null == xUrlResolver) {
    throw new java.lang.Exception(
        "Error: UrlResolver service does not export XUnoUrlResolver interface");
}
// use the interface
Object remoteObject = xUrlResolver.resolve(
    "uno:socket,host=0,port=2002;urp;StarOffice.ServiceManager");
```

The object decides whether or not it returns the interface. You have encountered a bug if the object does not return an interface that is specified to be mandatory in a service. When the interface reference is retrieved, invoke a call on the reference according to the interface specification. You can follow this strategy with every service you instantiate at a service manager, leading to success.

With this method, you may not only get UNO objects through the service manager, but also by normal interface calls:

```
// Module com::sun::star::text
interface XTextRange: com::sun::star::uno::XInterface
{
    XText getText();
    XTextRange getStart();
    ....
};
```

The returned interface types are specified in the operations, so that calls can be invoked directly on the returned interface. Often, an object implementing multiple interfaces are returned, instead of an object implementing one certain interface.

You can then query the returned object for the other interfaces specified in the given service, here `com.sun.star.drawing.Text`.

UNO has a number of generic interfaces. For example, the interface `com.sun.star.frame.XComponentLoader`:

```
// module com::sun::star::frame
interface XComponentLoader: com::sun::star::uno::XInterface
{
    com::sun::star::lang::XComponent loadComponentFromURL( [in] string aURL,
            [in] string aTargetFrameName,
            [in] long nSearchFlags,
            [in] sequence<com::sun::star::beans::PropertyValue> aArgs )
        raises( com::sun::star::io::IOException,
                com::sun::star::lang::IllegalArgumentException );
};
```

It becomes difficult to find which interfaces are supported beside `XComponent`, because the kind of returned document (text, calc, draw, etc.) depends on the incoming URL.

These dependencies are described in the appropriate chapters of this manual.

Tools such as the `InstanceInspector` component is a quick method to find out which interfaces a certain object supports. The `InstanceInspector` component comes with the OpenOffice.org SDK that allows the inspection of a certain object at runtime. Do not rely on implementation details of certain objects. If an object supports more interfaces than specified in the service description, query the interface and perform calls. The code may only work for this distinct office version and not work with an update of the office!



Unfortunately, there may still be bugs in the service specifications. Please provide feedback about missing interfaces to *openoffice.org* to ensure that the specification is fixed and that you can rely on the support of this interface.

There are certain specifications a `queryInterface()` implementation must not violate:

- If `queryInterface()` on a specific object returned a valid interface reference for a given type, it *must* return a valid reference for any successive `queryInterface()` calls on this object for the same type.

- If `queryInterface()` on a specific object returned a null reference for a given type, it *must* always return a null reference for the same type.
- If `queryInterface()` on reference A returns reference B, `queryInterface()` on B for Type A *must* return interface reference A or calls made on the returned reference *must* be equivalent to calls made on reference A.
- If `queryInterface()` on a reference A returns reference B, `queryInterface()` on A and B for XInterface *must* return the same interface reference (object identity).

These specifications must not be violated because a UNO runtime environment may choose to cache `queryInterface()` calls. The rules are basically identical to the rules of QueryInterface in MS COM.

3.3.4 Properties

Properties are name-value pairs belonging to a service and determine the characteristics of an object in a service instance. Usually, properties are used for non-structural attributes, such as font, size or color of objects, whereas get and set methods are used for structural attributes like a parent or sub-object.

In almost all cases, `com.sun.star.beans.XPropertySet` is used to access properties by name. Other interfaces, for example, are `com.sun.star.beans.XPropertyAccess` which is used to set and retrieve all properties at once or `com.sun.star.beans.XMultiPropertySet` which is used to access several specified properties at once. This is useful on remote connections. Additionally, there are interfaces to access properties by numeric ID, such as `com.sun.star.beans.XFastPropertySet`.

The following example demonstrates how to query and change the properties of a given text document cursor using its `XPropertySet` interface:

```
    // get an XPropertySet, here the one of a text cursor
    XPropertySet xCursorProps = (XPropertySet)
            UnoRuntime.queryInterface(XPropertySet.class, mxDocCursor);

    // get the character weight property
    Object aCharWeight = xCursorProps.getPropertyValue("CharWeight");
    float fCharWeight = AnyConverter.toFloat(aCharWeight);
    System.out.println("before: CharWeight=" + fCharWeight);

    // set the character weight property to BOLD
    xCursorProps.setPropertyValue("CharWeight", new Float(com.sun.star.awt.FontWeight.BOLD));

    // get the character weight property again
    aCharWeight = xCursorProps.getPropertyValue("CharWeight");
    fCharWeight = AnyConverter.toFloat(aCharWeight);
    System.out.println("after: CharWeight=" + fCharWeight);
```

A possible output of this code could be:

```
before: CharWeight=100.0
after: CharWeight=150.0
```



The sequence of property names must be sorted.

The following example deals with multiple properties at once:

```
// get an XMultiPropertySet, here the one of the first paragraph
XEnumerationAccess xEnumAcc = (XEnumerationAccess) UnoRuntime.queryInterface(
    XEnumerationAccess.class, mxDocText);
XEnumeration xEnum = xEnumAcc.createEnumeration();
Object aPara = xEnum.nextElement();
XMultiPropertySet xParaProps = (XMultiPropertySet) UnoRuntime.queryInterface(
    XMultiPropertySet.class, aPara);
```

```
// get three property values with a single UNO call
String[] aNames = new String[3];
aNames[0] = "CharColor";
aNames[1] = "CharFontName";
aNames[2] = "CharWeight";
Object[] aValues = xParaProps.getPropertyValues(aNames);

// print the three values
System.out.println("CharColor=" + AnyConverter.toLong(aValues[0]));
System.out.println("CharFontName=" + AnyConverter.toString(aValues[1]));
System.out.println("CharWeight=" + AnyConverter.toFloat(aValues[2]));
```

Properties can be assigned flags to determine a specific behavior of the property, such as read-only, bound, constrained or void. Possible flags are specified in `com.sun.star.beans.PropertyAttribute`. Read-only properties cannot be set. Bound properties broadcast changes of their value to registered listeners and constrained properties veto changes to these listeners.

Properties might have a status specifying where the value comes from. See `com.sun.star.beans.XPropertyState`. The value determines if the value comes from the object, a style sheet or if it cannot be determined at all. For example, in a multi-selection with multiple values within this selection.

The following example shows how to find out status information about property values:

```
    // get an XPropertySet, here the one of a text cursor
    XPropertySet xCursorProps = (XPropertySet) UnoRuntime.queryInterface(
        XPropertySet.class, mxDocCursor);

    // insert "first" in NORMAL character weight
    mxDocText.insertString(mxDocCursor, "first ", true);
    xCursorProps.setPropertyValue("CharWeight", new Float(com.sun.star.awt.FontWeight.NORMAL));

    // append "second" in BODL characer weight
    mxDocCursor.collapseToEnd();
    mxDocText.insertString(mxDocCursor, "second", true);
    xCursorProps.setPropertyValue("CharWeight", new Float(com.sun.star.awt.FontWeight.BOLD));

    // try to get the character weight property of BOTH words
    mxDocCursor.gotoStart(true);
    try {
        Object aCharWeight = xCursorProps.getPropertyValue("CharWeight");
        float fCharWeight = AnyConverter.toFloat(aCharWeight );
        System.out.println("CharWeight=" + fCharWeight);
    } catch (NullPointerException e) {
        System.out.println("CharWeight property is NULL");
    }

    // query the XPropertState interface of the cursor properties
    XPropertyState xCursorPropsState = (XPropertyState) UnoRuntime.queryInterface(
        XPropertyState.class, xCursorProps);

    // get the status of the character weight property
    PropertyState eCharWeightState = xCursorPropsState.getPropertyState("CharWeight");
    System.out.print("CharWeight property state has ");
    if (eCharWeightState == PropertyState.AMBIGUOUS_VALUE)
        System.out.println("an ambiguous value");
    else
        System.out.println("a clear value");
```

The property state of character weight is queried for a string like this:

first **second**

And the output is:

```
CharWeight property is NULL
CharWeight property state has an ambiguous value
```

The description of properties available for a certain object is given by `com.sun.star.beans.XPropertySetInfo`. Multiple objects can share the same property information for their description. This makes it easier for introspective caches that are used in scripting languages where the properties are accessed directly, without directly calling the methods of the interfaces mentioned above.

This example shows how to find out which properties an object provides using `com.sun.star.beans.XPropertySetInfo`:

```
try {
    // get an XPropertySet, here the one of a text cursor
    XPropertySet xCursorProps = (XPropertySet)UnoRuntime.queryInterface(
        XPropertySet.class, mxDocCursor);

    // get the property info interface of this XPropertySet
    XPropertySetInfo xCursorPropsInfo = xCursorProps.getPropertySetInfo();

    // get all properties (NOT the values) from XPropertySetInfo
    Property[] aProps = xCursorPropsInfo.getProperties();
    int i;
    for (i = 0; i < aProps.length; ++i) {
        // number of property within this info object
        System.out.print("Property #"  + i);

        // name of property
        System.out.print(": Name<" + aProps[i].Name);

        // handle of property (only for XFastPropertySet)
        System.out.print("> Handle<" + aProps[i].Handle);

        // type of property
        System.out.print("> " + aProps[i].Type.toString());

        // attributes (flags)
        System.out.print(" Attributes<");
        short nAttribs = aProps[i].Attributes;
        if ((nAttribs & PropertyAttribute.MAYBEVOID) != 0)
            System.out.print("MAYBEVOID|");
        if ((nAttribs & PropertyAttribute.BOUND) != 0)
        System.out.print("BOUND|");
        if ((nAttribs & PropertyAttribute.CONSTRAINED) != 0)
            System.out.print("CONSTRAINED|");
        if ((nAttribs & PropertyAttribute.READONLY) != 0)
            System.out.print("READONLY|");
        if ((nAttribs & PropertyAttribute.TRANSIENT) != 0)
            System.out.print("TRANSIENT|");
        if ((nAttribs & PropertyAttribute.MAYBEAMBIGUOUS ) != 0)
            System.out.print("MAYBEAMBIGUOUS|");
        if ((nAttribs & PropertyAttribute.MAYBEDEFAULT) != 0)
            System.out.print("MAYBEDEFAULT|");
        if ((nAttribs & PropertyAttribute.REMOVEABLE) != 0)
            System.out.print("REMOVEABLE|");
        System.out.println("0>");
    }
} catch (Exception e) {
    // If anything goes wrong, give the user a stack trace
    e.printStackTrace(System.out);
}
```

The following is an example output for the code above. The output shows the names of the text cursor properties, and their handle, type and property attributes. The handle is not unique, since the specific object does not implement com.sun.star.beans.XFastPropertySet, so proper handles are not needed here.

```
Using default connect string: socket,host=localhost,port=8100
Opening an empty Writer document
Property #0: Name<BorderDistance> Handle<93> Type<long> Attributes<MAYBEVOID|0>
Property #1: Name<BottomBorder> Handle<93> Type<com.sun.star.table.BorderLine> Attributes<MAYBEVOID|0>
Property #2: Name<BottomBorderDistance> Handle<93> Type<long> Attributes<MAYBEVOID|0>
Property #3: Name<BreakType> Handle<81> Type<com.sun.star.style.BreakType> Attributes<MAYBEVOID|0>

...

Property #133: Name<TopBorderDistance> Handle<93> Type<long> Attributes<MAYBEVOID|0>
Property #134: Name<UnvisitedCharStyleName> Handle<38> =Type<string> Attributes<MAYBEVOID|0>
Property #135: Name<VisitedCharStyleName> Handle<38> Type<string> Attributes<MAYBEVOID|0>
```

In some cases properties are used to specify the options in a sequence of `com.sun.star.beans.PropertyValue`. See `com.sun.star.view.PrintOptions` or `com.sun.star.document.MediaDescriptor` for examples properties in sequences. These are not accessed by the methods mentioned above, but by accessing the sequence specified in the language binding.

This example illustrates how to deal with sequences of property values:

```
// create a sequence of PropertyValue
```

```
PropertyValue[] aArgs = new PropertyValue[2];

// set name/value pairs (other fields are irrelevant here)
aArgs[0] = new PropertyValue();
aArgs[0].Name = "FilterName";
aArgs[0].Value = "HTML (StarWriter)";
aArgs[1] = new PropertyValue();
aArgs[1].Name = "Overwrite";
aArgs[1].Value = Boolean.TRUE;

// use this sequence of PropertyValue as an argument
// where a service with properties but witouth any interfaces is specified
com.sun.star.frame.XStorable xStorable = (com.sun.star.frame.XStorable) UnoRuntime.queryInterface(
    com.sun.star.frame.XStorable.class, mxDoc);
xStorable.storeAsURL("file:///tmp/devmanual-test.html", aArgs);
```

Usually the properties supported by an object, as well as their type and flags are fixed over the lifetime of the object. There may be exceptions. If the properties can be added and removed externally, the interface `com.sun.star.beans.XPropertyContainer` has to be used. In this case, the fixed `com.sun.star.beans.XPropertySetInfo` changes its supplied information over the lifetime of the object. Listeners for such changes can register at `com.sun.star.beans.XPropertyChangeListener`.



If you use a component from other processes or remotely, try to adhere to the rule to use `com.sun.star.beans.XPropertyAccess` and `com.sun.star.beans.XMultiPropertySet` instead of having a separate call for each single property.

The following diagram shows the relationship between the property-related interfaces.

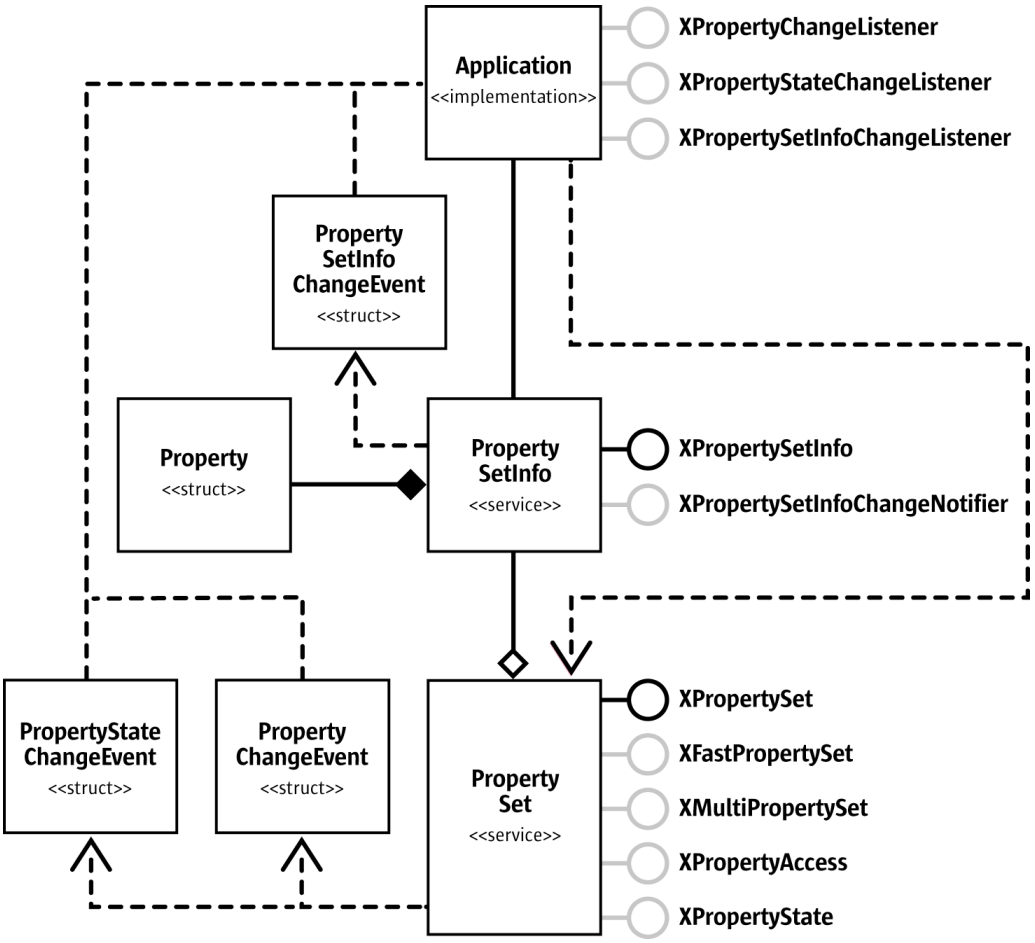


Illustration 19: Properties

3.3.5 Collections and Containers

Collections and *containers* are concepts for objects that contain multiple sub-objects where the number of sub-objects is usually not predetermined. While the term *collection* is used when the sub-objects are implicitly determined by the collection itself, the term *container* is used when it is possible to add new sub-objects and remove existing sub-objects explicitly. Thus, containers add methods like `insert()` and `remove()` to the collection interfaces.

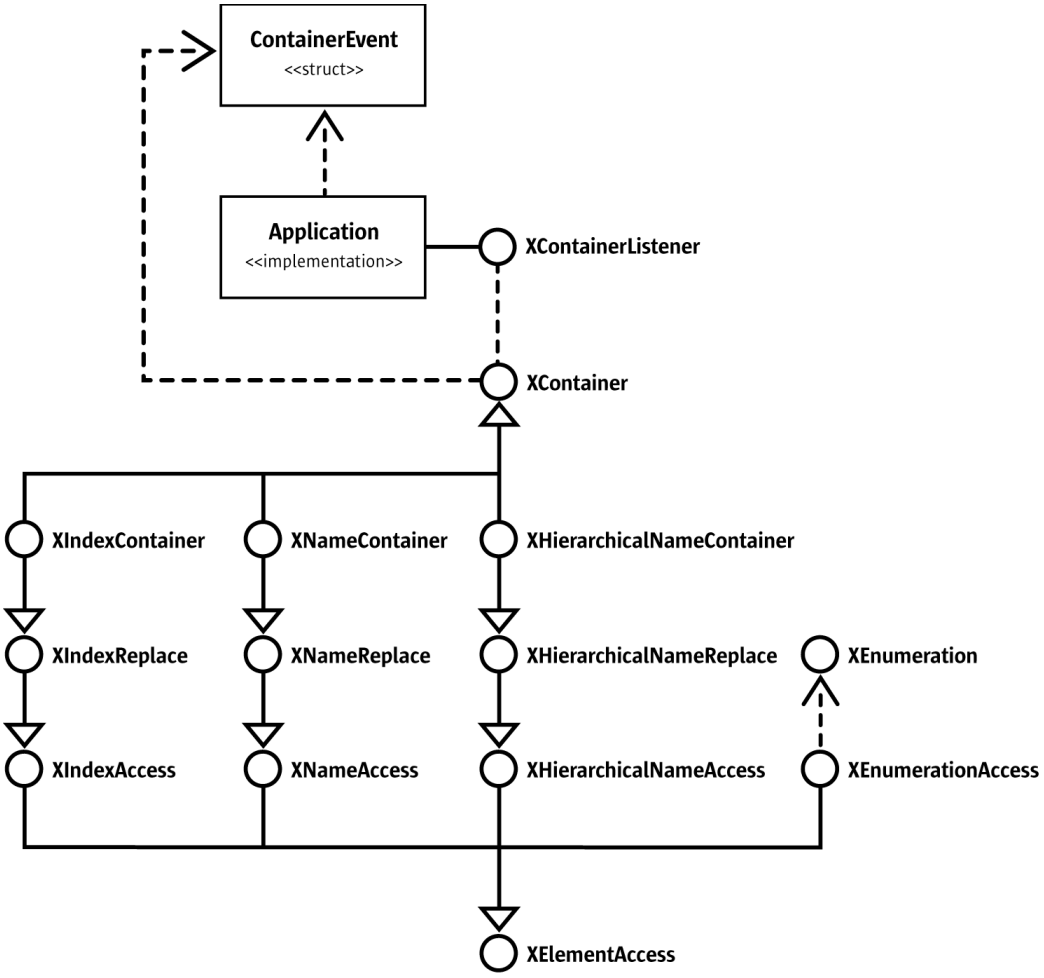


Illustration 20: Interfaces in `com.sun.star.container`

In general, the OpenOffice.org API collection and container interfaces contain any type that can be represented by the UNO type `any`. However, many container instances can be bound to a specific type or subtypes of this type. This is a runtime and specification agreement, and cannot be checked at runtime.

The base interface for collections is `com.sun.star.container.XElementAccess` that determines the types of the sub-object, if they are determined by the collection, and the number of contained sub-objects. Based on `XElementAccess`, there are three main types of collection interfaces:

- `com.sun.star.container.XIndexAccess`
  Offers direct access to the sub-objects by a subsequent numeric index beginning with 0.
- `com.sun.star.container.XNameAccess`
  Offers direct access to the sub-objects by a unique name for each sub object.

- `com.sun.star.container.XEnumerationAccess`
  Creates uni-directional iterators that enumerate all sub-objects in an undefined order.

`com.sun.star.container.XIndexAccess` is extended by `com.sun.star.container.XIndexReplace` to replace existing sub-objects by index, and `com.sun.star.container.XIndexContainer` to insert and remove sub-objects. You can find the same similarity for `com.sun.star.container.XNameAccess` and other specific collection types.

All containers support `com.sun.star.container.XContainer` that has interfaces to register `com.sun.star.container.XContainerListener` interfaces. This way it is possible for an application to learn about insertion and removal of sub-objects in and from the container.



The `com.sun.star.container.XIndexAccess` is appealing to programmers because in most cases, it is easy to implement. But this interface should only be implemented if the collection really is indexed.

Refer to the module `com.sun.star.container` in the API reference for details about collection and container interfaces.

The following examples demonstrate the usage of the three main collection interfaces. First, we iterate through an indexed collection. The index always starts with 0 and is continuous:

```
// get an XIndexAccess interface from the collection
XIndexAccess xIndexAccess = (XIndexAccess) UnoRuntime.queryInterface(
    XIndexAccess.class, mxCollection);

// iterate through the collection by index
int i;
for (i = 0; i < xIndexAccess.getCount(); ++i) {
    Object aSheet = xIndexAccess.getByIndex(i);
    Named xSheetNamed = (XNamed) oRuntime.queryInterface(XNamed.class, aSheet);
    System.out.println("sheet #" + i + " is named '" + xSheetNamed.getName() + "'");
}
```

Our next example iterates through a collection with named objects. The element names are unique within the collection and case sensitive.

```
// get an XNameAccess interface from the collection
XNameAccess xNameAccess = (XNameAccess) UnoRuntime.queryInterface(XNameAccess.class, mxCollection);

// get the list of names
String[] aNames = xNameAccess.getElementNames();

// iterate through the collection by name
int i;
for (i = 0; i < aNames.length; ++i) {
    // get the i-th object as a UNO Any
    Object aSheet = xNameAccess.getByName( aNames[i] );

    // get the name of the sheet from its XNamed interface
    XNamed xSheetNamed = (XNamed) UnoRuntime.queryInterface(XNamed.class, aSheet);
    System.out.println("sheet '" + aNames[i] + "' is #" + i);
}
```

The next example shows how we iterate through a collection using an enumerator. The order of the enumeration is undefined. It is only defined that all elements are enumerated. The behavior is undefined, if the collection is modified after creation of the enumerator.

```
// get an XEnumerationAccess interface from the collection
XEnumerationAccess xEnumerationAccess = (XEnumerationAccess) UnoRuntime.queryInterface(
    XEnumerationAccess.class, mxCollection );

// create an enumerator
XEnumeration xEnum = xEnumerationAccess.createEnumeration();

// iterate through the collection by name
while (xEnum.hasMoreElements()) {
    // get the next element as a UNO Any
    Object aSheet = xEnum.nextElement();

    // get the name of the sheet from its XNamed interface
    XNamed xSheetNamed = (XNamed) UnoRuntime.queryInterface(XNamed.class, aSheet);
    System.out.println("sheet '" + xSheetNamed.getName() + "'");
}
```

For an example showing the use of containers, see *7.4.1 Text Documents - Overall Document Features - Styles* where a new style is added into the style family `ParagraphStyles`.

3.3.6 Event Model

Events are a well known concept in graphical user interface (GUI) models, although they can be used in many contexts. The purpose of events is to notify an application about changes in the components used by the application. In a GUI environment, for example, an event might be the click on a button. Your application might be registered to this button and thus be able to execute certain code when this button is clicked.

The OpenOffice.org event model is similar to the JavaBeans event model. Events in OpenOffice.org are, for example, the creation or activation of a document, as well as the change of the current selection within a view. Applications interested in these events can register handlers (listener interfaces) that are called when the event occurs. Usually these listeners are registered at the object container where the event occurs or to the object itself. These listener interfaces are named `X...Listener`.

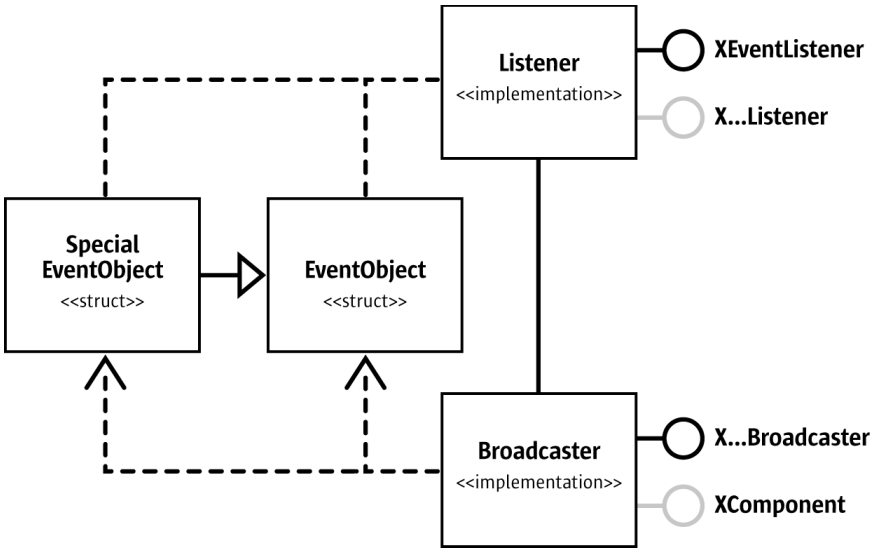


Illustration 21

Event listeners are subclasses of `com.sun.star.lang.XEventListener` that receives one event by itself, the deletion of the object to which the listener is registered. On this event, the listener has to unregister from the object, otherwise it would keep it alive with its interface reference counter.



Important! Implement the method `disposing()` to unregister at the object you are listening to and release all other references to this object.

Many event listeners can handle several events. If the events are generic, usually a single callback method is used. Otherwise, multiple callback methods are used. These methods are called with at least one argument: `com.sun.star.lang.EventObject`. This argument specifies the source of the event, therefore, making it possible to register a single event listener to multiple objects and still know where an event is coming from. Advanced listeners might get an extended version of this event descriptor struct.

3.3.7 Exception Handling

UNO uses *exceptions* as a mechanism to propagate errors from the called method to the caller. This error mechanism is preferred instead of error codes (as in MS COM) to allow a better separation of the error handling code from the code logic. Furthermore, Java, C++ and other high-level programming languages provide an exception handling mechanism, so that this can be mapped easily into these languages.

In IDL, an exception is a structured container for data, comparable to IDL structs. Exceptions cannot be passed as a return value or method argument, because the IDL compiler does not allow this. They can be specified in `raise` clauses and transported in an `any`. There are two kinds of exceptions, *user-defined* exceptions and *runtime* exceptions.

User-Defined Exceptions

The designer of an interface should declare exceptions for every possible error condition that might occur. Different exceptions can be declared for different conditions to distinguish between different error conditions.

The implementation may throw the specified exceptions and exceptions derived from the specified exceptions. The implementation must not throw unspecified exceptions, that is, the implementation must not throw an exception if no exception is specified. This applies to all exceptions except for RuntimeExceptions, described later.

When a user-defined exception is thrown, the object should be left in the state it was in before the call. If this cannot be guaranteed, then the exception specification must describe the state of the object. Note that this is not recommended.

Every UNO IDL exception must be derived from `com.sun.star.uno.Exception`, whether directly or indirectly. Its UNO IDL specification looks like this:

```
module com {  module sun {  module star {  module uno {
exception Exception
{
    string Message;
    com::sun::star::uno::XInterface Context;
};
}; }; }; };
```

The exception has two members:

- The message should contain a detailed readable description of the error (in English), which is useful for debugging purposes, though it cannot be evaluated at runtime. There is currently no concept of having localized error messages.
- The Context member should contain the object that initially threw the exception.

The following .IDL file snippet shows a method with a proper exception specification and proper documentation.

```
module com {  module sun {  module star {  module beans {

interface XPropertySet: com::sun::star::uno::XInterface
{
    ...
    /** @returns
        the value of the property with the specified name.

        @param PropertyName
            This parameter specifies the name of the property.

        @throws UnknownPropertyException
            if the property does not exist.

        @throws com::sun::star::uno::lang::WrappedTargetException
            if the implementation has an internal reason for the
```

```
            exception. In this case the original exception
            is wrapped into that WrappedTargetException.
    */
    any getPropertyValue( [in] string PropertyName )
        raises( com::sun::star::beans::UnknownPropertyException,
                com::sun::star::lang::WrappedTargetException );
    ...
};

}; }; }; };
```

Runtime Exceptions

Throwing a runtime exception signals an exceptional state. Runtime exceptions and exceptions derived from runtime exceptions cannot be specified in the raise clause of interface methods in IDL.

These are a few reasons for throwing a runtime exception are:

- The connection of an underlying interprocess bridge has broken down during the call.
- An already disposed object is called (see `com.sun.star.lang.XComponent` and the called object cannot fulfill its specification because of its disposed state.
- A method parameter was passed in an explicitly forbidden manner. For instance, a null interface reference was passed as a method argument where the specification of the interface explicitly forbids this.

Every UNO call may throw a `com.sun.star.uno.RuntimeException`, except acquire and release. This is independent of how many calls have been completed successfully. Every caller should ensure that its own object is kept in a consistent state even if a call to another object replied with a runtime exception. The caller should also ensure that no resource leaks occur in these cases. For example, allocated memory, file descriptors, etc.

If a runtime exception occurs, the caller does not know if the call has been completed successfully or not. The `com.sun.star.uno.RuntimeException` is derived from `com.sun.star.uno.Exception`. Note, that in the Java UNO binding, the `com.sun.star.uno.Exception` is derived from java.lang.Exception, while the `com.sun.star.uno.RuntimeException` is directly derived from java.lang.RuntimeException.

A common misuse of the runtime exception is to reuse it for an exception that was forgotten during interface specification. This should be avoided under all circumstances. Consider, defining a new interface.

An exception should not be misused as a new kind of programming flow mechanism. It should always be possible that during a session of a program, no exception is thrown. If this is not the case, the interface design should be reviewed.

Good Exception Handling

This section provides tips on exception handling strategies. Under certain circumstances, the code snippets we call bad below might make sense, but often they do not.

- *Do not throw exceptions with empty messages*

Often, especially in C++ code where you generally do not have a stack trace, the message within the exception is the only method that informs the caller about the reason and origin of the exception. The message is important, especially when the exception comes from a generic interface where all kinds of UNO exceptions can be thrown.

When writing exceptions, put descriptive text into them. To transfer the text to another exception, make sure to copy the text.

- *Do not catch exceptions without handling them*

Many people write helper functions to simplify recurring coding tasks. However, often code will be written like the following:

```
// Bad example for exception handling
public static void insertIntoCell( XPropertySet xPropertySet ) {
    [...]
    try {
        xPropertySet.setPropertyValue("CharColor",new Integer(0));
    } catch (Exception e) {
    }
}
```

This code is ineffective, because the error is hidden. The caller will never know that an error has occurred. This is fine as long as test programs are written or to try out certain aspects of the API (although even test programs should be written correctly). Exceptions must be addressed because the compiler can not perform correctly. In real applications, handle the exception.

The appropriate solution depends on the appropriate handling of exceptions. The following is the minimum each programmer should do:

```
// During early development phase, this should be at least used instead
public static void insertIntoCell(XPropertySet xPropertySet) {
    [...]
    try {
        xPropertySet.setPropertyValue("CharColor",new Integer(0));
    } catch (Exception e) {
        e.dumpStackTrace();
    }
}
```

The code above dumps the exception and its stack trace, so that a message about the occurrence of the exception is received on `stderr`. This is acceptable during development phase, but it is insufficient for deployed code. Your customer does not watch the `stderr` window.

The level where the error can be handled must be determined. Sometimes, it would be better not to catch the exception locally, but further up the exception chain. The user can then be informed of the error through dialog boxes. Note that you can even specify exceptions on the `main()` function:

```
// this is how the final solution could look like
public static void insertIntoCell(XPropertySet xPropertySet) throws UnknownPropertyException,
        PropertyVetoException, IllegalArgumentException, WrappedTargetException {
    [...]
    xPropertySet.setPropertyValue("CharColor",new Integer(0));
}
```

As a general rule, if you cannot recover from an exception in a helper function, let the caller determine the outcome. Note that you can even throw exceptions at the `main()` method.

3.3.8 Lifetime of UNO Objects

The UNO component model has a strong impact on the lifetime of UNO objects, in contrast to CORBA, where object lifetime is completely unspecified. UNO uses the same mechanism as Microsoft COM by handling the lifetime of objects by reference counting.

Each UNO runtime environment defines its own specification on lifetime management. While in C++ UNO, each object maintains its own reference count. Java UNO uses the normal Java garbage collector mechanism. The UNO core of each runtime environment needs to ensure that it upholds the semantics of reference counting towards other UNO environments.

The last paragraph of this chapter explains the differences between the lifetime of Java and C++ objects in detail.

acquire() and release()

Every UNO interface is derived from `com.sun.star.uno.XInterface`:

```
// module com::sun::star::uno
interface XInterface
{
    any queryInterface( [in] type aType );
    [oneway] void acquire();
    [oneway] void release();
};
```

UNO objects must maintain an internal reference counter. Calling `acquire()` on a UNO interface increases the reference count by one. Calling `release()` on UNO interfaces decreases the reference count by one. If the reference count drops to zero, the UNO object may be destroyed. Destruction of an object is sometimes called *death* of an object or that the object dies. The reference count of an object must always be non-negative.

Once `acquire()` is called on the UNO object, there is a *reference* or a *hard reference* to the object, as opposed to a weak reference. Calling `release()` on the object is often called *releasing* or *clearing* the reference.

The UNO object does not export the state of the reference count, that is, `acquire()` and `release()` do not have return values. Generally, the UNO object should not make any assumptions on the concrete value of the reference count, except for the transition from one to zero.

The invocation of a method is allowed first when `acquire()` has been called before. For every call to `acquire()`, there must be a corresponding release call, otherwise the object leaks.



Note: The UNO Java binding encapsulates `acquire()` and `release()` in the `UnoRuntime.queryInterface()` call. The same applies to the `Reference<>` template in C++. As long as the interface references are obtained through these mechanisms, `acquire()` and `release()` do not have to be called in your programs.

The XComponent Interface

A central problem of reference counting systems is cyclic references. Assume Object A keeps a reference on object B and B keeps a direct or indirect reference on object A. Even if all the external references to A and B are released, the objects are not destroyed, which results in a resource leak.

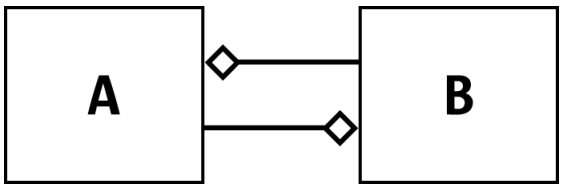


Illustration 22: Cyclic Reference



In general, a Java developer does not have to be concerned about this kind of issue, as the garbage collector algorithm detects ring references. However, in the UNO world one never knows, whether object A and object B really live in the same Java virtual machine. If they do, the ring reference is really garbage collected. If they do not, the object leaks, because the Java VM is not able to inspect the object outside of the VM for its references.

In UNO, the developer must explicitly decide when to the break cyclic references. To support this concept, the interface `com.sun.star.lang.XComponent` exists. When an `XComponent` is disposed of, it can inform other objects that have expressed interest to be notified.

```
// within the module com::sun::star::lang
// when dispose() is called, previously added XEventListeners are notified
interface XComponent: com::sun::star::uno::XInterface
{
     void dispose();
     void addEventListener( [in] XEventListener xListener );
     void removeEventListener( [in] XEventListener aListener );
};

// An XEventListener is notified by calling its disposing() method
interface XEventListener: com::sun::star::uno::XInterface
{
   void disposing( [in] com::sun::star::lang::EventObject Source );
};
```

Other objects can add themselves as `com.sun.star.lang.XEventListener` to an `XComponent`. When the `dispose()` method is called, the object notifies all `XEventListeners` through the `disposing()` method and releases all interface references, thus breaking the cyclic reference.

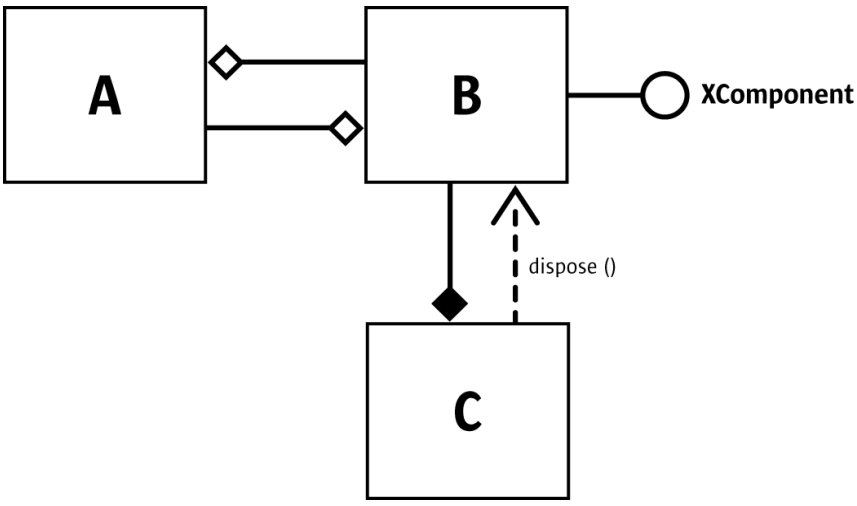


Illustration 23: Object C calls dispose() on XComponent of Object B

A disposed object is unable to comply with its specification, so it is necessary to ensure that an object is not disposed of before calling it. UNO uses an *owner/user* concept for this purpose. Only the owner of an object is allowed to call dispose and there can only be one owner per object. The owner is always free to dispose of the object. The user of an object knows that the object may be disposed of at anytime. The user adds an event listener to discover when an object is being disposed. When the user is notified, the user releases the interface reference to the object. In this case, the user should not call `removeEventListener()`, because the disposed object releases the reference to the user.



One major problem of the owner/user concept is that there always must be someone who calls `dispose()`. This must be considered at the design time of the services and interfaces, and be specified explicitly.

This solves the problem described above. However, there are a few conditions which still have to be met.

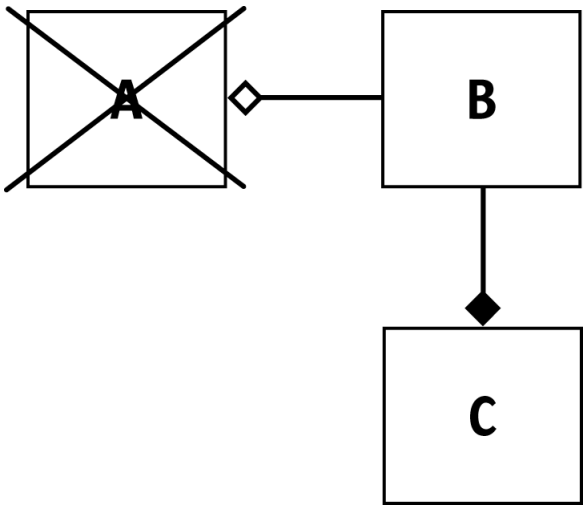


Illustration 24: B releases all interface references, which leads to destruction of Object A, which then releases its reference to B, thus the cyclic reference is broken.

If an object is called while it is disposed of, it should behave passively. For instance, if `removeListener()` is called, the call should be ignored. If methods are called while the object is no longer able to comply with its interface specification, it should throw a `com.sun.star.lang.DisposedException`, derived from `com.sun.star.uno.RuntimeException`. This is one of the rare situations in which an implementation should throw a `RuntimeException`. The situation described above can always occur in a multithreaded environment, even if the caller has added an event listener to avoid calling objects which were disposed of by the owner.

The owner/user concept may not always be appropriate, especially when there is more than one possible owner. In these cases, there should be no owner but only users. In a multithreaded scenario, `dispose()` might be called several times. The implementation of an object should be able to cope with such a situation.

The `XComponent` implementation should always notify the `disposing()` listeners that the object is being destroyed, not only when `dispose()` is called, but when the object is deleted. When the object is deleted, the reference count of the object drops to zero. This may happen when the listeners do not hold a reference on the broadcaster object.

The `XComponent` does not have to be implemented when there is only one owner and no further users.

Children of the XEventListener Interface

The `com.sun.star.lang.XEventListener` interface is the base for all listener interfaces . This means that not only `XEventListeners`, but every listener must implement `disposing()`, and every broadcaster object that allows any kind of listener to register, must call `disposing()` on the listeners as soon as it dies. However, not every broadcaster is forced to implement the `XComponent` interface with the `dispose()` method, because it may define its own condition when it is disposed.

In a chain of broadcaster objects where every element is a listener of its predecessor and only the root object is an `XComponent` that is being disposed, all the other chain links must handle the `disposing()` call coming from their predecessor and call `disposing()` on their registered listeners.

Weak Objects and References

A strategy to avoid cyclic references is to use *weak references*. Having a weak reference to an object means that you can reestablish a hard reference to the object again if the object still exists, and there is another hard reference to it.

In the cyclic reference shown in illustration *12: Interfaces, services and implementation*, object B could be specified to hold a hard reference on object A, but object A only keeps a weak reference to B. If object A needs to invoke a method on B, it temporarily tries to make the reference hard. If this succeeds, it invokes the method and releases the hard reference afterwards.

To be able to create a weak reference on an object, the object needs to support it explicitly by exporting the `com.sun.star.uno.XWeak` interface. The illustration *13: TV System Specification* depicts the UNO mechanism for weak references.

When an object is assigned to a weak reference, the weak reference calls `queryAdapter()` at the original object and adds itself (with the `com.sun.star.uno.XReference` interface) as reference to the adapter.

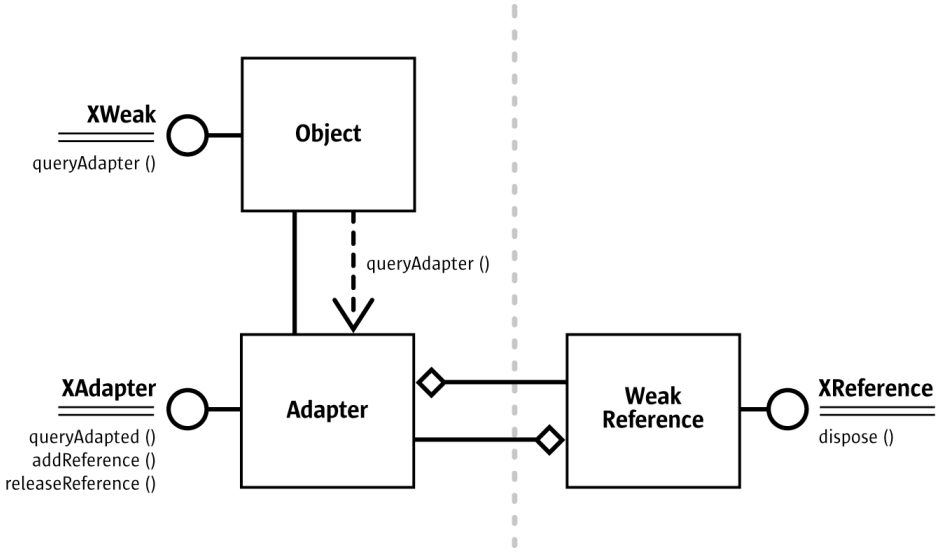


Illustration 25: The UNO weak reference mechanism

When a hard reference is established from the weak reference, it calls the `queryAdapted()` method at the `com.sun.star.uno.XAdapter` interface of the adapter object. When the original object is still alive, it gets a reference for it, otherwise a null reference is returned.

The adapter notifies the destruction of the original object to all weak references which breaks the cyclic reference between the adapter and weak reference.

4 Writing UNO Components describes the helper classes in C++ and Java that implement a `Xweak` interface and a weak reference..

Differences Between the Lifetime of C++ and Java Objects

Note: It is recommended that you read *3.4.2 Professional UNO - UNO Language Bindings - UNO C++ Binding* and *3.4.1 Professional UNO - UNO Language Bindings - Java Language Binding* for information on language bindings, and *4.6 Writing UNO Components - C++ Component* and *4.5.6 Writing UNO Components - Simple Component in Java - Storing the Service Manager for Further Use* about component implementation before beginning this section.

The implementation of the reference count specification is different in Java UNO and C++ UNO. In C++ UNO, every object maintains its own reference counter. When you implement a C++ UNO object, instantiate it, acquire it and afterwards release it, the destructor of the object is called immediately. The following example uses the standard helper class `::cppu::OWeakObject` and prints a message when the destructor is called. [SOURCE:ProfUNO/Lifetime/object_lifetime.cxx]

```
class MyOWeakObject : public ::cppu::OWeakObject
{
public:
    MyOWeakObject() { fprintf( stdout, "constructed\n" ); }
    ~MyOWeakObject() { fprintf( stdout, "destroyed\n" ); }
};
```

The following method creates a new `MyOWeakObject`, acquires it and releases it for demonstration purposes. The call to `release()` immediately leads to the destruction of `MyOWeakObject`. If the `Reference<>` template is used, you do not need to care about `acquire()` and `release()`.

```
void simple_object_creation_and_destruction()
{
    // create the UNO object
    com::sun::star::uno::XInterface * p = new MyOWeakObject();

    // acquire it
    p->acquire();

    // releast it
    fprintf( stdout, "before release\n" );
    p->release();
    fprintf( stdout, "after release\n" );
}
```

This piece of code produces the following output:

```
constructed
before release
destroyed
after release
```

Java UNO objects behave differently, because they are finalized by the garbage collector at a time of its choosing. `com.sun.star.uno.XInterface` has no methods in the Java UNO language binding, therefore no methods need to be implemented, although `MyUnoObject` implements `XInterface`: [SOURCE:ProfUNO/Lifetime/MyUnoObject.java]

```
class MyUnoObject implements com.sun.star.uno.XInterface {

    public MyUnoObject() {
    }

    void finalize() {
        System.out.println("finalizer called");
    }

    static void main(String args[]) throws java.lang.InterruptedException {
        com.sun.star.uno.XInterface a = new MyUnoObject();
        a = null;

        // ask the garbage collector politely
        System.gc();
        System.runFinalization();

        System.out.println("leaving");

        // It is java VM dependent, whether or not the finalizer was called
    }
}
```

The output of this code depends on the Java VM implementation. The output “finalizer called” is not a usual result. Be aware of the side effects when UNO brings Java and C++ together.

When a UNO C++ object is mapped to Java, a Java proxy object is created that keeps a hard UNO reference to the C++ object. The UNO core takes care of this. The Java proxy only clears the reference when it enters the `finalize()` method, thus the destruction of the C++ object is delayed until the Java VM collects some garbage.

When a UNO Java object is mapped to C++, a C++ proxy object is created that keeps a hard UNO reference to the Java object. Internally, the Java UNO bridge keeps a Java reference to the original Java object. When the C++ proxy is no longer used, it is destroyed immediately. The Java UNO bridge is notified and immediately frees the Java reference to the original Java object. When the Java object is finalized is dependent on the garbage collector.

When a Java program is connected to a running OpenOffice.org, the UNO objects in the office process are not destroyed until the garbage collector finalizes the Java proxies or until the interprocess connection is closed (see *3.3.1 Professional UNO - UNO Concepts - UNO Interprocess Connections*).

3.3.9 Object Identity

UNO guarantees if two object references are identical, that a check is performed and it always leads to a correct result, whether it be true or false. This is different from CORBA, where a return of false does not necessarily mean that the objects are different.

Every UNO runtime environment defines how this check should be performed. In Java UNO, there is a static `areSame()` function at the `com.sun.star.uno.UnoRuntime` class. In C++, the check is performed with the `Reference<>::operator == ()` function that queries both references for `XInterface` and compares the resulting `XInterface` pointers.

This has a direct effect in the API design. For instance, look at `com.sun.star.lang.XComponent`:

```
interface XComponent: com::sun::star::uno::XInterface
{
    void dispose();
    void addEventListener( [in] XEventListener xListener );
    void removeEventListener( [in] XEventListener aListener );
};
```

The method `removeEventListener()` that takes a listener reference, is logical if the implementation can check for object identity, otherwise it could not identify the listener that has to be removed. CORBA interfaces are not designed in this manner. They need an object ID, because object identity is not guaranteed.

3.4 UNO Language Bindings

This chapter documents the mapping of UNO to various programming languages or component models. These language bindings are sometimes called UNO Runtime Environment (URE). Each URE needs to fulfill the specifications given in the earlier chapters. The use of UNO services and interfaces are also explained in this chapter. Refer to *4 Writing UNO Components* for information about the implementation of UNO objects.

Each chapter provides detail information for the following topics:

- Mapping of all UNO types to the programming language types.
- Mapping of the UNO exception handling to the programming language.
- Mapping of the XInterface features (querying interfaces, object lifetime, object identity).
- Bootstrapping of a service manager.

Other programming language specific material (like core libraries in C++ UNO).

C++, Java, OpenOffice.org Basic and all languages supporting MS OLE automation on the win32 platform are currently supported. In future, the UNO component model may extend the number of supported language bindings.

3.4.1 Java Language Binding

The Java language binding gives developers the choice of using Java or UNO components for client programs. A Java program can access components written in other languages and built with a different compiler, as well as remote objects, because of the seamless interaction of UNO bridges.

Java delivers a rich set of classes that can be used within client programs or component implementations. However, when it comes to interaction with other UNO objects, use UNO interfaces, because only those are known to the bridge and can be mapped into other environments.

To control the office from a client program, the client needs a Java 1.3 installation, a free socket port, and the following jar files *jurt.jar, jut.jar, javaunohelper.jar, ridl.jar, classes.jar and sandbox.jar.* A Java installation on the server-side is not necessary. A step-by-step description is given in the chapter *2 First Steps*

When using Java components, the office is installed with Java support. Also make sure that Java is enabled: there is a switch that can be set to achieve this in the **Tools - Options - OpenOffice.org - Security** dialog. All necessary jar files should have been installed during the OpenOffice.org setup. A detailed explanation can be found in the chapter *4.5.6 Writing UNO Components - Simple Component in Java - Storing the Service Manager for Further Use.*

The Java UNO Runtime is documented in the Java UNO Reference which can be found in the OpenOffice.org Software development Kit (SDK) or on *udk.openoffice.org.*

Getting a Service Manager

Office objects that provide the desired functionality are required when writing a client application that accesses the office. The root of all these objects is the service manager component, therefore clients need to instantiate it. Service manager runs in the office process, therefore office must be running first when you use Java components that are instantiated by the office. In a client-server scenario, the office has to be launched directly. The interprocess communication uses a remote protocol that can be provided as a command-line argument to the office:

```
soffice -accept=socket,host=localhost,port=8100;urp;
```

The client obtains a reference to the global service manager of the office (the server) using a local `com.sun.star.bridge.UnoUrlResolver`. The global service manager of the office is used to get objects from the other side of the bridge. In this case, an instance of the `com.sun.star.frame.Desktop` is acquired:

```
import com.sun.star.uno.XComponentContext;
import com.sun.star.comp.helper.Bootstrap;
import com.sun.star.lang.XMultiComponentFactory;
import com.sun.star.bridge.XUnoUrlResolver;
import com.sun.star.beans.XPropertySet
import com.sun.star.uno.UnoRuntime;

XComponentContext xcomponentcontext = Bootstrap.createInitialComponentContext(null);

// initial serviceManager
XMultiComponentFactory xLocalServiceManager = xcomponentcontext.getServiceManager();

// create a connector, so that it can contact the office
Object xUrlResolver = xLocalServiceManager.createInstanceWithContext(
    "com.sun.star.bridge.UnoUrlResolver", xcomponentcontext);

XUnoUrlResolver urlResolver = (XUnoUrlResolver) UnoRuntime.queryInterface(
```

```
    XUnoUrlResolver.class, xUrlResolver);

Object initialObject = urlResolver.resolve(
    "uno:socket,host=localhost,port=8100;urp;StarOffice.ServiceManager");

XMultiComponentFactory xOfficeFactory = (XMultiComponentFactory) UnoRuntime.queryInterface(
    XMultiComponentFactory.class, initialObject);

// retrieve the component context as property (it is not yet exported from the office)
// Query for the XPropertySet interface.
XPropertySet xProperySet = (XPropertySet) UnoRuntime.queryInterface(
    XPropertySet.class, xOfficeFactory);

// Get the default context from the office server.
Object oDefaultContext = xProperySet.getPropertyValue("DefaultContext");

// Query for the interface XComponentContext.
XComponentContext xOfficeComponentContext = (XComponentContext) UnoRuntime.queryInterface(
    XComponentContext.class, oDefaultContext);

// now create the desktop service
// NOTE: use the office component context here!
Object oDesktop = xOfficeFactory.createInstanceWithContext(
                "com.sun.star.frame.Desktop", xOfficeComponentContext);
```

As the example shows, a local service manager is created through the `com.sun.star.comp.helper.Bootstrap` class (a Java UNO runtime class). Its implementation provides a service manager that is limited in the number of services it can create. The names of these services are:

```
com.sun.star.lang.ServiceManager
com.sun.star.lang.MultiServiceFactory
com.sun.star.loader.Java
com.sun.star.loader.Java2
com.sun.star.bridge.UnoUrlResolver
com.sun.star.bridge.BridgeFactory
com.sun.star.connection.Connector
com.sun.star.connection.Acceptor
```

They are sufficient to establish a remote connection and obtain the fully featured service manager provided by the office.



The local service manager could create other components, but this is only possible if the service manager is provided with the respective factories during runtime. An example that shows how this works can be found in the implementation of the Bootstrap class in the project javaunohelper.
There is also a service manager that uses a registry database to locate services. It is implemented by the class com.sun.star.comp.helper.RegistryServiceFactory in the project javaunohelper. However, the implementation uses a native registry service manager instead of providing a pure Java implementation.

Handling Interfaces

The service manager is created in the server process and the Java UNO remote bridge ensures that its `XInterface` is transported back to the client. A Java proxy object is constructed that can be used by the client code. This object is called the *initial object* , because it is the first object created by the bridge. When another object is obtained through this object, then the bridge creates a new proxy. For instance, if a function is called that returns an interface. That is, the original object is actually running in the server process (the office) and calls to the proxy are forwarded by the bridge. Not only interfaces are converted, but function arguments, return values and exceptions.

The Java bridge maps objects on a per-interface basis, that is, in the first step only the interface is converted that is returned by a function described in the API reference. For example, if you have the service manager and use it to create another component, you initially get a `com.sun.star.uno.XInterface`:

```
XInterface xint= (XInterface) serviceManager.createInstance("com.sun.star.bridge.OleObjectFactory");
```

You know from the service description that the OleObjectFactory implements a `com.sun.star.lang.XMultiServiceFactory` interface. However, you cannot cast the object or call the interface function on the object, since the object is only a proxy for just one interface, `XInterface`. Therefore, you have to use a mechanism that is provided with the Java bridge that generates proxy objects on demand. For example:

```
XMultiServiceFactory xfac = (XMultiServiceFactory) UnoRuntime.queryInterface(
    XMultiServiceFactory.class, xint);
```

If `xint` is a proxy, then `queryInterface()` hands out another proxy for `XMultiServiceFactory` provided that the original object implements it. Interface proxies can be used as arguments in function calls on other proxy objects. For example:

```
// client side
// obj is a proxy interface and returns another interface through its func() method
XSomething ret = obj.func();

// anotherObject is a proxy interface, too. Its method func(XSomething arg)
// takes the interface ret obtained from obj
anotherObject.func(ret);
```

In the server process, the *obj* object would receive the original *ret* object as a function argument.

It is also possible to have Java components on the client side. As well, they can be used as function arguments, then the bridge would set up proxies for them in the server process.

Not all language elements of UNO IDL have a corresponding language element in Java. For example, there are no structs and all-purpose out parameters. Refer to *3.4.1 Professional UNO - UNO Language Bindings - Java Language Binding - Type Mappings* for how those elements are mapped.

Interface handling normally involves the ability of `com.sun.star.uno.XInterface` to acquire and release objects by reference counting. In Java, the programmer does not bother with `acquire()` and `release()`, since the Java UNO runtime automatically acquires objects on the server side when `com.sun.star.uno.UnoRuntime.queryInterface()` is used. Conversely, when the Java garbage collector deletes your references, the Java UNO runtime releases the corresponding office objects. If a UNO object is written in Java, no reference counting is used to control its lifetime. The garbage collector takes that responsibility.

Sometimes it is necessary to find out if two interfaces belong to the same object. In Java, you would compare the references with the equality operator '==='. This works as long as the interfaces refer to a local Java object. Often the interfaces are proxies and the real objects reside in a remote process. There can be several proxies that belong to the same object, because objects are bridged on a per-interface basis. Those proxies are Java objects and comparing their references would not establish them as parts of the same object. To determine if interfaces are part of the same object, use the method `areSame()` at the `com.sun.star.uno.UnoRuntime` class:

```
static public boolean areSame(Object object1, Object object2)
```

Type Mappings

Mapping of Simple Types

The following table shows the mapping of IDL basic types to the corresponding Java types.

Users should be careful when using unsigned types in Java, since there is no support for unsigned types in the Java language. A user is responsible for the conversion of large unsigned IDL type values as signed values in Java.

IDL	Java
boolean	boolean
short	short
unsigned short	short
long	int
usigned long	int
hyper	long
unsigned hyper	long
float	float
double	double
char	char
byte	byte
string	java.lang.String
any	java.lang.Object/com.sun.star.uno.Any
type	com.sun.star.uno.Type
void	void

Mapping of Any

There is a dedicated `com.sun.star.uno.Any` type, but it is not always used. An `any` in the API reference is represented by a `java.lang.Object` in Java UNO. An `Object` reference can be used to refer to all possible Java objects. This does not work with primitive types, but if you need to use them as an `any`, there are Java wrapper classes available that allow primitive types to be used as objects. Also, a Java `Object` always brings along its type information by means of an instance of `java.lang.Class`. Therefore a variable declared as :

```
Object ref;
```

can be used with all objects and its type information is available by calling:

```
ref.getClass();
```

Those qualities of `Object` are sufficient to replace the `Any` in most cases. Even Java interfaces generated from IDL interfaces do not contain `Any`s, instead `Object` references are used in place of `Any`s. Cases where an explicit `Any` is needed to not loose information contain unsigned integral types, all interface types except the basic `XInterface`, and the void type.



However, implementations of those interfaces must be able to deal with real `Any`s that can also be passed by means of `Object` references.

To facilitate the handling of the `Any` type, use the `com.sun.star.uno.AnyConverter` class. It is documented in the Java UNO reference. The following list sums up its methods:

```
static boolean isArray(java.lang.Object object)
static boolean isBoolean(java.lang.Object object)
static boolean isByte(java.lang.Object object)
static boolean isChar(java.lang.Object object)
static boolean isDouble(java.lang.Object object)
static boolean isFloat(java.lang.Object object)
static boolean isInt(java.lang.Object object)
static boolean isLong(java.lang.Object object)
static boolean isObject(java.lang.Object object)
static boolean isShort(java.lang.Object object)
static boolean isString(java.lang.Object object)
static boolean isType(java.lang.Object object)
static boolean isVoid(java.lang.Object object)
static java.lang.Object toArray(java.lang.Object object)
static boolean toBoolean(java.lang.Object object)
static byte toByte(java.lang.Object object)
static char toChar(java.lang.Object object)
```

```
    static double toDouble(java.lang.Object object)
    static float toFloat(java.lang.Object object)
    static int toInt(java.lang.Object object)
    static long toLong(java.lang.Object object)
    static java.lang.Object toObject(Type type, java.lang.Object object)
    static short toShort(java.lang.Object object)
    static java.lang.String toString(java.lang.Object object)
    static Type toType(java.lang.Object object)
```

The Java `com.sun.star.uno.Any` is needed in situations when the type needs to be specified explicitly. Assume there is a C++ component with an interface function which is declared in UNO IDL as:

```
//UNO IDL
void foo(any arg);
```

The corresponding C++ implementation could be:

```
void foo(const Any& arg)
{
    const Type& t = any.getValueType();
    if (t == getCppuType((const Reference<XReference>*) 0))
    {
        Reference<XReference> myref = *reinterpret_cast<const Reference<XReference>*>(any.getValue());
        ...
    }
}
```

In the example, the any is checked if it contains the expected interface. If is does, it is assigned accordingly. If the any contained a different interface, a query would be performed for the expected interface. If the function is called from Java, then an interface has to be supplied that is an object. That object could implement several interfaces and the bridge would use the basic `XInterface`. If this is not the interface that is expected, then the C++ implementation has to call `queryInterface` to obtain the desired interface. In a remote scenario, those `queryInterface()` calls could lead to a noticeable performance loss. If you use a Java `Any` as a parameter for `foo()`, the intended interface is sent across the bridge.

Preserving UNO Type Information for Complex Types

In C++ UNO, all necessary type information is described by the type `Type`. In Java, type information is mapped to the Java type `Class`, but some information described in IDL is lost. The Java mapping for the complex types (interface, struct, exception) creates an additional public static final member array of type `com.sun.star.lib.uno.typelib.TypeInfo` named `UNOTYPEINFO` to describe this information. This array can be filled with objects of the following types:

- `MethodTypeInfo`
  To describe the attributes of a method, is it `oneway`, or `const` and if the return type is `unsigned`.
- `ParameterTypeInfo`
  To describe if the parameter type is `unsigned`, and if the direction is `inout` or `out`.
- `AttributeTypeInfo`
  To describe if the type is `unsigned` and if the attribute is `readonly`.
- `MemberTypeInfo`
  To describe if the type is `unsigned`

Note Only these definitions are maintained in UNOTYPEINFO. This additional type information and the information from `Class` is used by the Java UNO runtime to handle the type during transport over a remote connection or conversion to another object model.

All generated types (interface, struct, enum, exception) have another public static member of type Object UNORUNTIMEDATA. This member is reserved for internal use by the UNO runtime.

Mapping of Sequence

Sequence types are mapped to a Java array of the Java type that corresponds to the element types of the original IDL sequence.

- An IDL `sequence<long>` is mapped to `int[]`
- An IDL `sequence< sequence <long> >` is mapped to `int[][]`

Mapping of Module

An IDL module is mapped to a Java package with the same name. All IDL type declarations within the module are mapped to corresponding Java class or interface declarations within the generated package. IDL declarations not enclosed in any modules are mapped into the Java global scope.

Example:

An IDL `module org {...}` is mapped to `package org;` ...

Mapping of Interface

An IDL interface is mapped to a Java interface with the same name as the IDL interface type. If an IDL interface inherits another interface, the Java interface extends the appropriate Java interface.

Mapping of Method Parameters

In Java there are special conditions concerning the value `null` for parameters and return values, and concerning `out` and `inout` parameters. It is common for Java that arguments or return values which are objects can be `null`. Since UNO interfaces, sequences, structs and strings are mapped to Java objects (sequence is mapped to an array which is a special kind of object), the respective method arguments or return values could be `null`. But UNO allows only interface values to be passed as `null` values. If a UNO interface function has parameters, `in`, `inout` or `out` parameters, or a return value of type sequence, struct or string, then the respective values of the Java method must not be `null`. The example below uses a struct `FooStruct` in an interface `XFoo` to show how to use empty parameters and return values, and how to use `out` and `inout` parameters.

```
//UNO IDL
struct FooStruct
{
    long nval;
    string strval;
};

interface XFoo: com.sun.star.uno.XInterface
{
    string funcOne( [in] string value);
    FooStruct funcTwo( [inout] FooStruct value);
    sequence<byte> funcThree([out] sequence <byte> value);
};
```

IDL `in` parameters that call-by-value semantics are mapped to normal Java actual parameters. The result of IDL operations is returned as the result of the corresponding Java method. IDL `out` and `inout` parameters that implement call-by-reference semantics are mapped to arrays of the appropriate types. The type is determined according to the mappings defined in this document. The arrays contain one element, that is, the length of the array is `1`. Therefore, the Java interface for the IDL interface `XFoo` would look:

```
// Java
public interface XFoo extends com.sun.star.uno.XInterface {
    public String funcOne(String value);
    public FooStruct funcTwo(FooStruct[] value);
    public byte[] funcThree(byte[][] value);
    ...
```

```
}
```

This is how `FooStruct` would be mapped to Java:

```
// Java
public class FooStruct {
    public int nval;
    public String strval;

    // default constructor
    public FooStruct() {
      strval="";
    }

    public FooStruct(int _nval, String _strval) {
        nval = _nval;
        strval = _strval;
    }

    // extra type information
    ...
}
```

When providing a value as an `inout` parameter, the caller has to write the input value into the element at index `0` of the array. When the function returns, the value at index `0` reflects the output value, which may be a new value, modified input value, or unmodified input value. The object `obj` implements `XFoo`:

```
// calling the interface in Java
obj.funcOne(null);    // error
obj.funcOne("");      // OK

FooStruct[] inoutstruct= new FooStruct[1];
obj.funcTwo(inoutstruct);          //error, inoutstruct[0] = null

inoutstruct[0]= new FooStruct();   // now we initialise inoutstruct[0]
obj.funcTwo(inoutstruct);          // inoutstruct[0] is valid now
```

When a method receives an argument that is an `out` parameter, it has to provide a value that has to be put at index null of the array.

```
// method implementations of interface XFoo
public String funcOne(String value) {
  // param value always != null otherwise it is a bug of the caller!
  return null; //error
  // instead
  // return "";
}

public FooStruct funcTwo(FooStruct[] value) {
  value[0] = null; //error
  // instead
  // value[0] = new FooStruct();
  return null; // error
  // instead
  // return new FooStruct();
}

public byte[] funcThree(byte[][] value) {
  value[0]= null; //error
  // instead
  // value[0]= new byte[0];
  return null; //error
  // instead
  // return new byte[0];
}
```

Exceptions specified in UNO IDL are mapped to normal Java `throws` statements. Any method may throw a `com.sun.star.uno.RuntimeException`, therefore a `RuntimeException` never has to be specified explicitly in UNO IDL.

```
module com {  module sun {  module star {  module registry {


interface XImplementationRegistration: com::sun::star::uno::XInterface
{
    void registerImplementation(
            [in] string aImplementationLoader,
                [in] string aLocation,
                [in] com::sun::star::registry::XSimpleRegistry xReg )
            raises( com::sun::star::registry::CannotRegisterImplementationException );
```

```
    boolean revokeImplementation(
        [in] string aLocation,
        [in] com::sun::star::registry::XSimpleRegistry xReg );

    sequence getImplementations(
        [in] string aImplementationLoader,
        [in] string aLocation );

    sequence checkInstantiation( [in] string implementationName );
};
```

is mapped to:

```
package com.sun.star.registry;

public interface XImplementationRegistration extends com.sun.star.uno.XInterface {
    // Methods
    public void registerImplementation(/*IN*/String aImplementationLoader,
            /*IN*/String aLocation, /*IN*/XSimpleRegistry xReg)
        throws CannotRegisterImplementationException, com.sun.star.uno.RuntimeException;
    public boolean revokeImplementation(/*IN*/String aLocation, /*IN*/XSimpleRegistry xReg)
        throws com.sun.star.uno.RuntimeException;
    public String[] getImplementations(/*IN*/String aImplementationLoader, /*IN*/String aLocation)
        throws com.sun.star.uno.RuntimeException;
    public String[] checkInstantiation(/*IN*/String implementationName)
        throws com.sun.star.uno.RuntimeException;

    // static Member
    public static final com.sun.star.lib.uno.typeinfo.TypeInfo UNOTYPEINFO[] = {
        new com.sun.star.lib.uno.typeinfo.MethodTypeInfo("registerImplementation", 0, 0),
        new com.sun.star.lib.uno.typeinfo.ParameterTypeInfo("xReg", "registerImplementation", 2,
            com.sun.star.lib.uno.typeinfo.TypeInfo.INTERFACE),
        new com.sun.star.lib.uno.typeinfo.MethodTypeInfo("revokeImplementation", 1, 0),
        new com.sun.star.lib.uno.typeinfo.ParameterTypeInfo("xReg", "revokeImplementation", 1,
            com.sun.star.lib.uno.typeinfo.TypeInfo.INTERFACE),
        new com.sun.star.lib.uno.typeinfo.MethodTypeInfo("getImplementations", 2, 0),
        new com.sun.star.lib.uno.typeinfo.MethodTypeInfo("checkInstantiation", 3, 0)
     };

    public static Object UNORUNTIMEDATA = null;
}
```

Mapping of Structs

An IDL struct is mapped to a Java class with the same name as the struct type. Each member of the IDL struct is mapped to a public instance variable with the same type and name. The class also provides a default constructor which initializes all members with default values, and a constructor which takes values for all struct members. If a struct inherits from another struct, the generated class extends the class of the inherited struct. The default constructor only initializes the complex type members. The member constructor has all fields of the extended class and its own fields as parameters.

```
module com {  module sun {  module star {  module chart {

struct ChartDataChangeEvent: com::sun::star::lang::EventObject
{
    com::sun::star::chart::ChartDataChangeType Type;
    short StartColumn;
    short EndColumn;
    short StartRow;
    short EndRow;
};


}; }; }; };
```

is mapped to:

```
package com.sun.star.chart;

public class ChartDataChangeEvent extends com.sun.star.lang.EventObject {
    //instance variables
    public ChartDataChangeType Type;
    public short StartColumn;
    public short EndColumn;
    public short StartRow;
    public short EndRow;

    //constructors
    public ChartDataChangeEvent() {
```

```
        Type = com.sun.star.chart.ChartDataChangeType.getDefault();
    }

    public ChartDataChangeEvent(java.lang.Object _Source, ChartDataChangeType _Type,
            short _StartColumn, short _EndColumn, short _StartRow, short _EndRow ) {
        super( _Source );
        Type = _Type;
        StartColumn = _StartColumn;
        EndColumn = _EndColumn;
        StartRow = _StartRow;
        EndRow = _EndRow;
    }

    public static final com.sun.star.lib.uno.typeinfo.TypeInfo UNOTYPEINFO[] = {
        new com.sun.star.lib.uno.typeinfo.MemberTypeInfo("Type", 0, 0),
        new com.sun.star.lib.uno.typeinfo.MemberTypeInfo("StartColumn", 1, 0),
        new com.sun.star.lib.uno.typeinfo.MemberTypeInfo("EndColumn", 2, 0),
        new com.sun.star.lib.uno.typeinfo.MemberTypeInfo("StartRow", 3, 0),
        new com.sun.star.lib.uno.typeinfo.MemberTypeInfo("EndRow", 4, 0)
        };
}
```

Mapping of Exceptions

An IDL exception is mapped to a Java class with the same name as the exception type.

There are two UNO exceptions that are the base for all other exceptions. These are the `com.sun.star.uno.Exception` and `com.sun.star.uno.RuntimeException` that are inherited by all other exceptions. The corresponding exceptions in Java inherit from Java exceptions:

```
//UNO IDL
module com {  module sun {  module star {  module uno {
exception Exception
{
    string Message;
    com::sun::star::uno::XInterface Context;
};
}; }; }; };

module com { module sun { module star { module uno {
exception RuntimeException
{
    string Message;
    com::sun::star::uno::XInterface Context;
};
}; }; }; };
```

The `com.sun.star.uno.Exception` in Java:

```
package com.sun.star.uno;

public class Exception extends java.lang.Exception {
    // instance variables
    public java.lang.Object Context;

    // constructors
    public Exception() {
    }

    public Exception(String _Message) {
        super (_Message);
    }

    public Exception(String _Message, java.lang.Object _Context) {
        super (_Message);
        Context = _Context;
    }

    public static final com.sun.star.lib.uno.typeinfo.TypeInfo UNOTYPEINFO[] = {
        new com.sun.star.lib.uno.typeinfo.MemberTypeInfo("Context", 0,
            com.sun.star.lib.uno.typeinfo.TypeInfo.INTERFACE)
        };
}
```

The `com.sun.star.uno.RuntimeException` in Java:

```
package com.sun.star.uno;

public class RuntimeException extends java.lang.RuntimeException {
    // instance variables
    public java.lang.Object Context;

    // constructors
    public RuntimeException() {
    }

    public RuntimeException(String _Message) {
        super (_Message);
    }

    public RuntimeException(String _Message, java.lang.Object _Context) {
        super( _Message );
        Context = _Context;
    }

    public static final com.sun.star.lib.uno.typeinfo.TypeInfo UNOTYPEINFO[] = {
        new com.sun.star.lib.uno.typeinfo.MemberTypeInfo("Context", 0,
            com.sun.star.lib.uno.typeinfo.TypeInfo.INTERFACE)
        };
}
```

As shown, the `Message` member has no direct counterpart in the respective Java class. Instead, the constructor argument `_Message` is used to initialize the base class which is a Java exception. The message is accessible through the inherited `getMessage()` method. All other members of the IDL exceptions are mapped to public instance variables with the same type and name. A generated Java exception class always has a default constructor that initializes all members with default values, and a constructor which takes values for all instance variables.

If an exception inherits from another exception, the generated class extends the class of the inherited exception, and the constructor takes the arguments for all fields of the class and the base classes.

Mapping of Enums and Constants

An IDL `enum` is mapped to a Java `final class` with the same name as the enum type, derived from the class `com.sun.star.uno.Enum`. This base class declares a protected member to store the actual value, a protected constructor to initialize the value and a public `getValue()` method to get the actual value. The generated final class has a protected constructor and a public method `getDefault()` that returns an enum with the value of the first enum label as a default. For each IDL enum label, the class declares a public static member of the same type as the enum and is initialized with the defined value in IDL. The Java class for the enum has an additional public method `fromInt()` that which returns the enum with the specified value. The following IDL definition for `com.sun.star.uno.TypeClass`:

```
module com { module sun { star { module uno {
    enum TypeClass
    {
        INTERFACE,
        SERVICE,
        IMPLEMENTATION,
        STRUCT,
        TYPEDEF,
        ...
    };
}; }; }; };
```

is mapped to:

```
package com.sun.star.uno;

final public class TypeClass extends com.sun.star.uno.Enum {
    private TypeClass(int value) {
        super (value);
    }

    public static TypeClass getDefault() {
        return INTERFACE;
```

```
    }

    public static final TypeClass INTERFACE = new TypeClass(0);
    public static final TypeClass SERVICE = new TypeClass(1);
    public static final TypeClass IMPLEMENTATION = new TypeClass(2);
    public static final TypeClass STRUCT = new TypeClass(3);
    public static final TypeClass TYPEDEF = new TypeClass(4);
    ...

    public static TypeClass fromInt(int value) {
        switch (value) {
          case 0:
              return INTERFACE;
          case 1:
              return SERVICE;
          case 2:
              return IMPLEMENTATION;
          case 3:
              return STRUCT;
          case 4:
              return TYPEDEF;
          ...
        }
    }

    public static Object UNORUNTIMEDATA = null;
}
```

An IDL `const` named `USERFLAG`:

```
module example {
    const long USERFLAG = 1;
};
```

is mapped to:

```
package example;

public interface USERFLAG {
    public static final int value = (int)1L;
}
```

IDL `constants` groups are mapped to a public interface with the same name as the `constants` group. All `const` defined in this constant group are mapped to public static members of the interface with type and name of the `const` that holds the value.

An IDL constants group `User` containing three `const` values `FLAG1`, `FLAG2` and `FLAG3`:

```
module example {
    constants User
    {
        const long FLAG1 = 1;
        const long FLAG2 = 2;
        const long FLAG3 = 3;
    };
};
```

is mapped to:

```
package example;

public interface User {
    public static final int FLAG1 = (int)1L;
    public static final int FLAG2 = (int)2L;
    public static final int FLAG3 = (int)3L;
}
```

3.4.2 UNO C++ Binding

This chapter describes the UNO C++ language binding. It provides an experienced C++ programmer the first steps in UNO to establish UNO interprocess connections to a remote OpenOffice.org and to use its UNO objects.

Library Overview

Illustration 16: ComponentContext and the ServiceManager gives an overview about the core libraries of the UNO component model.

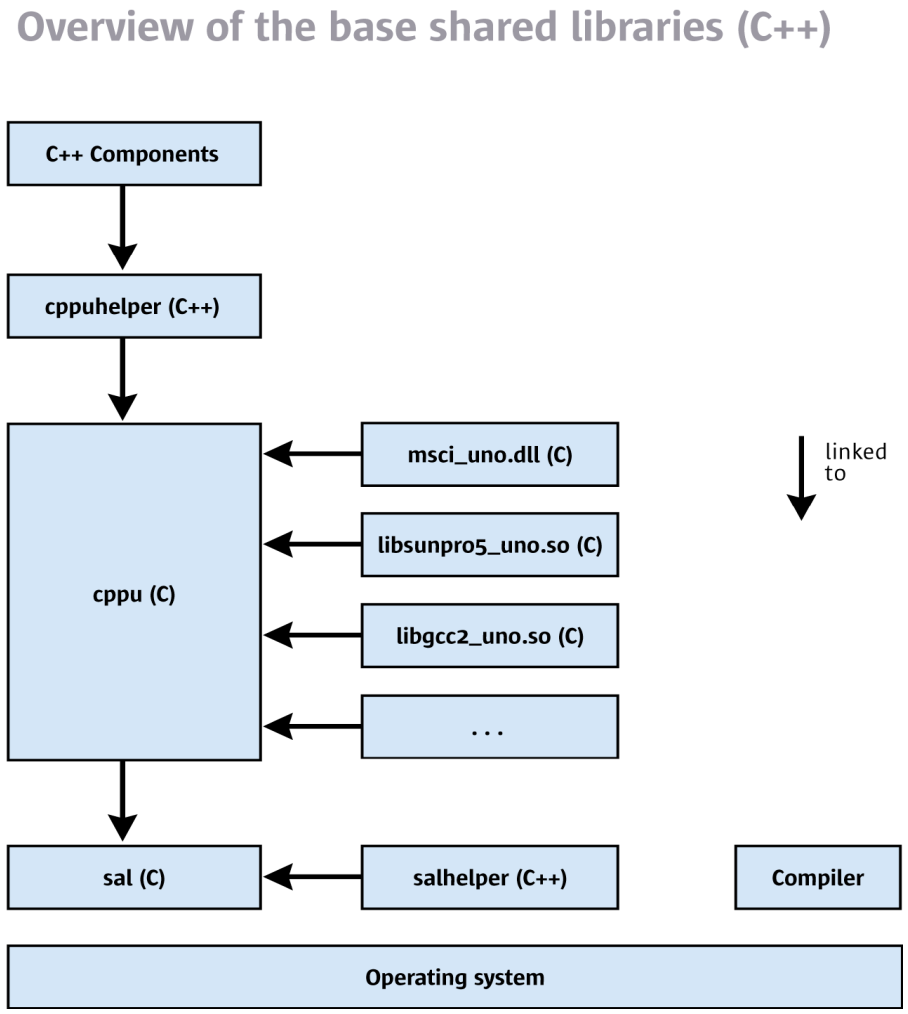


Illustration 26: Shared Libraries for C++ UNO

These shared libraries can be found in the *<officedir>/program* folder of your OpenOffice.org installation. The label (c) in the illustration above means C-linkage and (C++) means C++ linkage. For all libraries, a C++ compiler to build is required.

The basis for all UNO libraries is the sal library. It contains the *system abstraction layer* (sal) and additional runtime library functionality, but does not contain any UNO-specific information. The commonly used C-functions of the sal library can be accessed through C++ inline wrapper classes. This allows functions to be called from any other programming language, because most programming languages have some mechanism to call a C function.

The salhelper library is a small C++ library which offers additional runtime library functionality, that could not be implemented inline.

The cppu (C++ UNO) library is the core UNO library. It offers methods to access the UNO type library, and allows the creation, copying and comparing values of UNO data types in a generic manner. Moreover, all UNO bridges (= mappings + environments) are administered in this library.

The examples *msci_uno.dll*, *libsunpro5_uno.so* and *libgcc2_uno.so* are only examples for language binding libraries for certain C++ compilers.

The cppuhelper library is a C++ library that contains important base classes for UNO objects and functions to bootstrap the UNO core. C++ Components and UNO programs have to link the cppuhelper library.

All the libraries shown above will be kept compatible in all future releases of UNO. You will be able to build and link your application and component once, and run it with the current and later versions of OpenOffice.org.

System Abstraction Layer

C++ UNO client programs and C++ UNO components use the system abstraction layer (sal) for types, files, threads, interprocess communication, and string handling. The sal library offers operating system dependent functionality as C-functions. The aim is to minimize or to eliminate operating system dependent `#ifdef` in libraries above sal. Sal offers high performance access because sal is a thin layer above the API offered by each operating system.



In OpenOffice.org GUI APIs are encapsulated in the vcl library.

Sal exports only C-symbols. The inline C++ wrapper exists for convenience. Refer to the UNO C++ reference that is part of the OpenOffice.org SDK or in the References section of *udk.openoffice.org* to gain a full overview of the features provided by the sal library. In the following sections, the C++ wrapper classes will be discussed. The sal types used for UNO IDL types are discussed in section *3.4.2 Professional UNO - UNO Language Bindings - UNO C++ Binding - Type Mappings*. If you want to use them, look up the names of the appropriate include files in the C++ reference.

File Access

The classes listed below manage platform independent file access. They are C++ classes that call corresponding C functions internally.

- osl::FileBase
- osl::VolumeInfo
- osl::FileStatus
- osl::File
- osl::DirectoryItem
- osl::Directory

An unfamiliar concept is the use of absolute filenames throughout the whole API. In a multi-threaded program, the current working directory cannot be relied on, thus relative paths must be explicitly made absolute by the caller.

Threadsafe Reference Counting

The functions `osl_incrementInterlockedCount()` and `osl_decrementInterlockedCount()` in the global C++ namespace increase and decrease a 4-byte counter in a threadsafe manner. This is needed for reference counted objects. Many UNO APIs control object lifetime through refcounting.

Since concurrent incrementing the same counter does not increase the reference count reliably, these functions should be used. This is faster than using a mutex on most platforms.

Threads and Thread Synchronization

The class osl::Thread is meant to be used as a base class for your own threads. Overwrite the `run()` method.

The following classes are commonly used synchronization primitives:

osl::Mutex

- osl::Condition
- osl::Semaphore

Socket and Pipe

The following classes allow you to use interprocess communication in a platform independent manner:

- osl::Socket
- osl::Pipe

Strings

The classes rtl::OString (8-bit, encoded) and rtl::OUString (16-bit, UTF16) are the base-string classes for UNO programs. The strings store their data in a heap memory block. The string is refcounted and incapable of changing, thus it makes copying faster and creation is an expensive operation. An OUString can be created using the static function `OUString::createFromASCII()` or it can be constructed from an 8-bit string with encoding using this constructor:

```
OUString( const sal_Char * value,
        sal_Int32 length,
        rtl_TextEncoding encoding,
        sal_uInt32 convertFlags = OSTRING_TO_OUSTRING_CVTFLAGS );
```

It can be converted into an 8-bit string, for example, for `printf()` using the `rtl::OUStringToOString()` function that takes an encoding, such as RTL_TEXTENCODING_ASCII_US).

For fast string concatenation, the classes rtl::OStringBuffer and rtl::OUStringBuffer should be used, because they offer methods to concatenate strings and numbers. After preparing a new string buffer, use the `makeStringAndClear()` method to create the new `OUString` or `OString`. The following example illustrates this :

```
sal_Int32 n = 42;
double pi = 3.14159;

// create a buffer with a suitable size, rough guess is sufficient
// stringbuffer extends if necessary
OUStringBuffer buf( 128 );

// append an ascii string
buf.appendAscii( "pi ( here " );

// numbers can be simply appended
buf.append( pi );
// RTL_CONSTASCII_STRINGPARAM()
// lets the compiler count the stringlength, so this is more efficient than
// the above appendAscii call, where the length of the string must be calculated at
// runtime
buf.appendAscii( RTL_CONSTASCII_STRINGPARAM(" ) multiplied with " ) );
buf.append( n );
buf.appendAscii( RTL_CONSTASCII_STRINGPARAM(" gives ") );
buf.append( (double)( n * pi ) );
buf.appendAscii( RTL_CONSTASCII_STRINGPARAM( "." ) );

// now transfer the buffer into the string.
// afterwards buffer is empty and may be reused again !
```

```
    OUString string = buf.makeStringAndClear();

    // You could of course use the OStringBuffer directly to get an OString
    OString oString = rtl::OUStringToOString( string , RTL_TEXTENCODING_ASCII_US );

    // just to print something
    printf( "%s\n" ,oString.getStr() );
```

Establishing Interprocess Connections

Any language binding supported by UNO establishes interprocess connections using a local service manager to create the services necessary to connect to the office. Refer to chapter *3.3.1 Professional UNO - UNO Concepts - UNO Interprocess Connections* for additional information. The following client program connects to a running office and retrieves the `com.sun.star.lang.XMultiServiceFactory` in C++:
(ProfUNO/CppBinding/office_connect.cxx)

```
#include <stdio.h>

#include <cppuhelper/bootstrap.hxx>
#include <com/sun/star/bridge/XUnoUrlResolver.hpp>
#include <com/sun/star/lang/XMultiServiceFactory.hpp>

using namespace com::sun::star::uno;
using namespace com::sun::star::lang;
using namespace com::sun::star::bridge;
using namespace rtl;
using namespace cppu;

int main( )
{
    // create the initial component context
    Reference< XComponentContext > rComponentContext =
        defaultBootstrap_InitialComponentContext();

    // retrieve the service manager from the context
    Reference< XMultiComponentFactory > rServiceManager =
        rComponentContext->getServiceManager();

    // instantiate a sample service with the service manager.
    Reference< XInterface > rInstance =
        rServiceManager->createInstanceWithContext(
            OUString::createFromAscii("com.sun.star.bridge.UnoUrlResolver" ),
            rComponentContext );

    // Query for the XUnoUrlResolver interface
    Reference< XUnoUrlResolver > rResolver( rInstance, UNO_QUERY );

    if( ! rResolver.is() )
    {
        printf( "Error: Couldn't instantiate com.sun.star.bridge.UnoUrlResolver service\n" );
        return 1;
    }
    try
    {
        // resolve the uno-URL
        rInstance = rResolver->resolve( OUString::createFromAscii(
            "uno:socket,host=localhost,port=2002;urp;StarOffice.ServiceManager" ) );

        if( ! rInstance.is() )
        {
            printf( "StarOffice.ServiceManager is not exported from remote process\n" );
            return 1;
        }

        // query for the simpler XMultiServiceFactory interface, sufficient for scripting
        Reference< XMultiServiceFactory > rOfficeServiceManager (rInstance, UNO_QUERY);

        if( ! rOfficeServiceManager.is() )
        {
            printf( "XMultiServiceFactory interface is not exported\n" );
            return 1;
        }

        printf( "Connected sucessfully to the office\n" );
    }
    catch( Exception &e )
    {
        OString o = OUStringToOString( e.Message, RTL_TEXTENCODING_ASCII_US );
        printf( "Error: %s\n", o.pData->buffer );
```

```
        return 1;
    }
    return 0;
}
```

Type Mappings

Mapping of Simple Types

The following table provides a summary of the mappings from IDL types to C++ UNO types.

IDL type	Size [byte]	C++ type	Description
`void`	-	`void`	void
`byte`	1	`sal_Int8`	Signed 8-bit integer
`short`	2	`sal_Int16`	Signed 16-bit integer
`unsigned short`	2	`sal_uInt16`	Unsigned 16-bit integer
`signed long`	4	`sal_Int32`	Signed 32-bit integer
`unsigned long`	4	`sal_uInt32`	Unsigned 32-bit integer
`hyper`	8	`sal_Int64`	Signed 64-bit integer
`unsigned hyper`	8	`sal_uInt64`	Unsigned 64-bit integer
`float`	`sizeof (float)`	`float`	processor dependent: Intel, Sparc = IEEE float
`double`	`sizeof (double)`	`double`	processor dependent: Intel, Sparc = IEEE double
`boolean`	1	`sal_Bool { 0, 1 }`	8-bit unsigned char
`char`	2	`sal_Unicode`	16-bit unicode char
`string`	4	`rtl::OUString`	Unicode string
`type`	4	`com::sun::star::uno::Type`	Type descriptor

The basic integer types are all mapped to sal_x types, where x describes the bit length and sign of the simple type. The sal prefix is used to avoid name clashes with other libraries or applications.

A *string* is mapped to an rtl::OUString that is a reference counted, non-changing UTF-16 string. There are no 8-bit strings in UNO.

Mapping of Any

IDL type	Size [byte]	C++ type	Description
`any`	`sizeof (uno_Any)`	`com::sun::star::uno::Any`	universal type

The IDL `any` is mapped to com::sun::star::uno::Any. It holds an instance of an arbitrary UNO type. Only UNO types can be stored within the `any`, because the data from the type library are required for `any` handling.

A default constructed `Any` contains the void type and no value. You can assign a value to the `Any` using the operator `<<=` and retrieve a value using the operator `>>=`.

```
// default construct an any
Any any;

sal_Int32 n = 3;

// Store the value into the any
any <<= n;

// extract the value again
sal_Int32 n2;
any >>= n2;
assert( n2 == n );
assert( 3 == n2 );
```

The extraction operator `>>=` carries out widening conversions when no loss of data can occur, but data cannot be directed downward. If the extraction was successful, the operator returns `sal_True`, otherwise `sal_False`.

```
Any any;
sal_Int16 n = 3;
any <<= n;


sal_Int8 aByte = 0;
sal_Int16 aShort = 0;
sal_Int32 aLong = 0;

// this will succeed, conversion from int16 to int32 is OK.
assert( any >>= aLong );
assert( 3 == aLong );

// this will succeed, conversion from int16 to int16 is OK
assert( any >>= aShort );
assert( 3 == aShort

// the following two assertions will FAIL, because conversion
// from int16 to int8 may involve loss of data..

// Even if a downcast is possible for a certain value, the operator refuses to work
assert( any >>= aByte );
assert( 3 == aByte );
```

Instead of using the operator for extracting, you can also get a pointer to the data within the `Any`. This may be faster, but it is more complicated to use. With the pointer, care has to be used during casting and proper type handling, and the lifetime of the `Any` must exceed the pointer usage.

```
Any a = ...;
if( a.getTypeClass() == TypeClass_LONG && 3 ==  *(sal_Int32 *)a.getValue() )
{
}
```

You can also construct an Any from a pointer to a C++ UNO type that can be useful. For instance:

```
Any foo()
{
    sal_Int32 i = 3;
    if( ... )
        i = ..;
    return Any( &i, getCppuType( &i) );
}
```

Mapping of Interface

IDL type	Size [byte]	C++ type	Description
`Interface`	4	`com::sun::star::uno::Reference< interfacetype >`	Pointer to a refcounted interface

An IDL *interface reference* is mapped to the template class:

```
template< class t >
```

```
com::sun::star::uno::Reference< t >
```

The template is used to get a type safe interface reference, because only a correctly typed interface pointer can be assigned to the reference. The example below assigns an instance of the desktop service to the `rDesktop` reference:

```
// the xSMgr reference gets constructed somehow
{
    ...
    // construct a deskop object and acquire it
    Reference< XInterface > rDesktop = xSMgr->createInstance(
    OUString::createFromAscii("com.sun.star.frame.Desktop"));
    ...
    // reference goes out of scope now, release is called on the interface
}
```

The constructor of `Reference` calls `acquire()` on the interface and the destructor calls `release()` on the interface. These references are often called *smart pointers*. Always use the `Reference` template consistently to avoid reference counting bugs.

The `Reference` class makes it simple to invoke `queryInterface()` for a certain type:

```
// construct a deskop object and acquire it
Reference< XInterface > rDesktop = xSMgr->createInstance(
    OUString::createFromAscii("com.sun.star.frame.Desktop"));

// query it for the XFrameLoader interface
Reference< XFrameLoader > rLoader( rDesktop , UNO_QUERY );

// check, if the frameloader interface is supported
if( rLoader.is() )
{
    // now do something with the frame loader
    ...
}
```

The `UNO_QUERY` is a dummy parameter that tells the constructor to query the first constructor argument for the `XFrameLoader` interface. If the `queryInterface()` returns successfully, it is assigned to the `rLoader` reference. You can check if querying was successful by calling `is()` on the new reference.

Methods on interfaces can be invoked using the operator `->`:

```
xSMgr->createInstance(...);
```

The `operator()->()` returns the interface pointer without acquiring it, that is, without incrementing the refcount.



If you need the direct pointer to an interface for some purpose, you can also call `get()` at the reference class.

You can explicitly release the interface reference by calling `clear()` at the reference or by assigning a default constructed reference.

You can check if two interface references belong to the same object using the operator `==`.

Mapping of Sequence

An IDL *sequence* is mapped to:

```
template< class t >
com::sun::star::uno::Sequence< t >
```

The sequence class is a reference to a reference counted handle that is allocated on the heap.

The sequence follows a copy-on-modify strategy. If a sequence is about to be modified, it is checked if the reference count of the sequence is 1. If this is the case, it gets modified directly, otherwise a copy of the sequence is created that has a reference count of 1.

A sequence can be created with an arbitrary UNO type as element type, but do not use a non-UNO type. The full reflection data provided by the type library are needed for construction, destruction and comparison.

You can construct a sequence with an initial number of elements. Each element is default constructed.

```
{
    // create an integer sequence with 3 elements,
    // elements default to zero.
    Sequence< sal_Int32 > seqInt( 3 );

    // get a read/write array pointer (this method checks for
    // the refcount and does a copy on demand).
    sal_Int32 *pArray = seqInt.getArray();

    // if you know, that the refocunt is one
    // as in this case, where the sequence has just been
    // constructed, you could avoid the check,
    // which is a C-call overhead,
    // by writing sal_Int32 *pArray = (sal_Int32*) seqInt.getConstArray();

    // modify the members
    pArray[0] = 4;
    pArray[1] = 5;
    pArray[2] = 3;
}
```

You can also initialize a sequence from an array of the same type by using a different constructor. The new sequence is allocated on the heap and all elements are copied from the source.

```
{
    sal_Int32 sourceArray[3] = {3,5,3};

    // result is the same as above, but we initialize from a buffer.
    Sequence< sal_Int32 > seqInt( sourceArray , 3 );
}
```

Complex UNO types like structs can be stored within sequences, too:

```
{
    // construct a sequence of Property structs,
    // the structs are default constructed
    Sequence< Property > seqProperty(2);
    seqProperty[0].Name = OUString::createFromAscii( "A" );
    seqProperty[0].Handle = 0;
    seqProperty[1].Name = OUString::createFromAscii( "B" );
    seqProperty[1].Handle = 1;

    // copy construct the sequence (The refcount is raised)
    Sequence< Property > seqProperty2 = seqProperty;

    // access a sequence
    for( sal_Int32 i = 0 ; i < seqProperty.getLength() ; i ++ )
    {
        // Please NOTE : seqProperty.getArray() would also work, but
        //               it is more expensive, because a
        //               unnessecary copy construction
        //               of the sequence takes place.
        printf( "%d\n" , seqProperty.getConstArray()[i].Handle );
    }
}
```

The size of sequences can be changed using the `realloc()` method, which takes the new number of elements as a parameter. For instance:

```
// construct an empty sequence
Sequence < Any > anySequence;

// get your enumeration from somewhere
Reference< XEnumeration > rEnum = ...;

// iterate over the enumeration
while( rEnum->hasMoreElements() )
{
    anySequence.realloc( anySequence.getLength() + 1 );
    anySequence[anySequence.getLength()-1] = rEnum->nextElement();
}
```

The above code shows an enumeration is transformed into a sequence,using an inefficient method. The `realloc()` default constructs a new element at the end of the sequence. If the sequence is shrunk by `realloc`, the elements at the end are destroyed.

The sequence is meant as a transportation container only, therefore it lacks methods for efficient insertion and removal of elements. Use a C++ Standard Template Library vector as an intermediate container to manipulate a list of elements and finally copy the elements into the sequence.

Sequences of a specific type are a fully supported UNO type. There can also be a sequence of sequences. This is similar to a multidimensional array with the exception that each row may vary in length. For instance:

```
{
    sal_Int32 a[ ] = { 1,2,3 }, b[] = {4,5,6}, c[] = {7,8,9,10};
    Sequence< Sequence< sal_Int32 > > aaSeq ( 3 );
    aaSeq[0] = Sequence< sal_Int32 >( a , 3 );
    aaSeq[1] = Sequence< sal_Int32 >( b , 3 );
    aaSeq[2] = Sequence< sal_Int32 >( c , 4 );
}
```

is a valid sequence of sequence< sal_Int32>.

Mapping of Type

A **type** is mapped to `com::sun::star::uno::Type`. It holds the name of a type and the `com.sun.star.uno.TypeClass`. The type allows you to obtain a `com::sun::star::uno::TypeDescription` that contains all the information defined in the IDL. A UNO type object for a specific type using the overloaded `cppu::getCppuType()` function can be constructed:

```
// get the type of sal_Int32
Type intType = getCppuType( (sal_Int32 *) 0 );

// get the type of a string
Type stringType = getCppuType( (OUString * ) 0 );

// get the type of the XEnumeration interface
Type xenumerationType = getCppuType( (Reference<XEnumeration>*) 0 );
```

The above code is useful to write template functions. Some `getCppuType()` functions would be ambiguous. There are specialized functions: `getVoidCppuType()`, `getBooleanCppuType()`, `getCharCppuType()`to handle the ambiguous functions.

The functions are implemented inline and introduced by headers that have been generated from the type library.

Using Weak References

The C++ binding offers a method to hold UNO objects *weakly,* that is, not holding a hard reference to it. A hard reference prevents an object from being destroyed, whereas an object that is held weakly can be deleted anytime. The advantage of weak references is used to avoid cyclic references between objects.

An object must actively support weak references by supporting the `com.sun.star.uno.XWeak` interface. The concept is explained in detail in chapter *3.3.7 Professional UNO - UNO Concepts - Lifetime of UNO Objects.*

Weak references are often used for caching. For instance, if you want to reuse an existing object, but do not want to hold it forever to avoid cyclic references.

Weak references are implemented as a template class:

```
template< class t >
class com::sun::star::uno::WeakReference<t>
```

You can simply assign weak references to hard references and conversely. The following examples stress this:

```
// forward declaration of a function that
Reference< XFoo > getFoo();

int main()
{
    // default construct a weak reference.
    // this reference is empty
    WeakReference < XFoo > weakFoo;
    {
        // obtain a hard reference to an XFoo object
        Reference< XFoo > hardFoo = getFoo();
        assert( hardFoo.is() );

        // assign the hard reference to weak referencecount
        weakFoo = hardFoo;

        // the hardFoo reference goes out of scope. The object itself
        // is now destroyed, if no one else keeps a reference to it.
        // Nothing happens, if someone else still keeps a reference to it
    }

    // now make the reference hard again
    Reference< XFoo > hardFoo2 = weakFoo;


    // check, if this was successful
    if( hardFoo2.is() )
    {
        // the object is still alive, you can invoke calls on it again
        hardFoo2->foo();
    }
    else
    {
        // the objects has died, you can't do anything with it anymore.
    }
}
```

A call on a weak reference can not be invoked directly. Make the weak reference hard and check whether it succeeded or not. You never know if you will get the reference, therefore always handle both cases properly.

It is more expensive to use weak references instead of hard references. When assigning a weak reference to a hard reference, a mutex gets locked and some heap allocation may occur. When the object is located in a different process, at least one remote call takes place, meaning an overhead of approximately a millisecond.

The XWeak mechanism does not support notification at object destruction. For this purpose, objects must export XComponent and add `com.sun.star.lang.XEventListener`.

Exception Handling in C++

For throwing and catching of UNO exceptions, use the normal C++ exception handling mechanisms. Calls to UNO interfaces may only throw the `com::sun::star::uno::Exception` or derived exceptions. The following example catches every possible exception:

```
try
{
    Reference< XInterface > rInitialObject =
        xUnoUrlResolver->resolve( OUString::createFromAsci(
            "uno:socket,host=localhost,port=2002;urp;StarOffice.ServiceManager" ) );
}
catch( com::sun::star::uno::Exception &e )
{
    OString o = OUStringToOString( e.Message, RTL_TEXTENCODING_ASCII_US );
    printf( "An error occurred: %s\n", o.pData->buffer );
}
```

If you want to react differently for each possible exception type, look up the exceptions that may be thrown by a certain method. For instance the `resolve()` method in

`com.sun.star.bridge.XUnoUrlResolver` is allowed to throw three kinds of exceptions. Catch each exception type separately:

```
try
{
    Reference< XInterface > rInitialObject =
    xUnoUrlResolver->resolve( OUString::createFromAsci(
        "uno:socket,host=localhost,port=2002;urp;StarOffice.ServiceManager" ) );
}
catch( ConnectionSetupException &e )
{
    OString o = OUStringToOString( e.Message, RTL_TEXTENCODING_ASCII_US );
    printf( "%s\n", o.pData->buffer );
    printf( "couldn't access local resource ( possible security resons )\n" );
}
catch( NoConnectException &e )
{
    OString o = OUStringToOString( e.Message, RTL_TEXTENCODING_ASCII_US );
    printf( "%s\n", o.pData->buffer );
    printf( "no server listening on the resource\n" );
}
catch( IllegalArgumentException &e )
{
    OString o = OUStringToOString( e.Message, RTL_TEXTENCODING_ASCII_US );
    printf( "%s\n", o.pData->buffer );
    printf( "uno URL invalid\n" );
}
catch( RuntimeException & e )
{
    OString o = OUStringToOString( e.Message, RTL_TEXTENCODING_ASCII_US );
    printf( "%s\n", o.pData->buffer );
    printf( "an unknown error has occurred\n" );
}
```

When implementing your own UNO objects (see *4.6 Writing UNO Components - C++ Component*), throw exceptions using the normal C++ throw statement:

```
void MyUnoObject::initialize( const Sequence< Any > & args.getLength() ) throw( Exception )
{
    // we expect 2 elements in this sequence
    if( 2 != args.getLength() )
    {
        // create an error message
        OUStringBuffer buf;
        buf.appendAscii( "MyUnoObject::initialize, expected 2 args, got " );
        buf.append( args.getLength() );
        buf.append( "." );

        // throw the exception
        throw Exception( buf.makeStringAndClear() , *this );
    }
    ...
}
```

Note that only exceptions derived from `com::sun::star::uno::Exception` may be thrown at UNO interface methods. Other exceptions (for instance the C++ std::exception) cannot be bridged by the UNO runtime if the caller and called object are not within the same UNO Runtime Environment. Moreover, most current Unix C++ compilers, for instance gcc 3.0.x, do not compile code. During compilation, exception specifications are loosen in derived classes by throwing exceptions other than the exceptions specified in the interface that it is derived. Throwing unspecified exceptions leads to a std::unexpected exception and causes the program to abort on Unix systems.

3.4.3 OpenOffice.org Basic

OpenOffice.org Basic provides access to the OpenOffice.org API from within the office application. It hides the complexity of interfaces and simplifies the use of properties by making UNO objects look like Basic objects. It offers convenient Runtime Library (RTL) functions and special Basic properties for UNO. Furthermore, Basic procedures can be easily hooked up to GUI elements, such as menus, toolbar icons and GUI event handlers.

This chapter describes how to access UNO using the OpenOffice.org Basic scripting language. In the following sections, OpenOffice.org Basic is referred to as Basic.

Handling UNO Objects

Accessing UNO Services

UNO objects are used through their interface methods and properties. Basic simplifies this by mapping UNO interfaces and properties to Basic object methods and properties.

First, in Basic it is not necessary to distinguish between the different interfaces an object supports when calling a method. The following illustration shows an example of an UNO service that supports three interfaces:

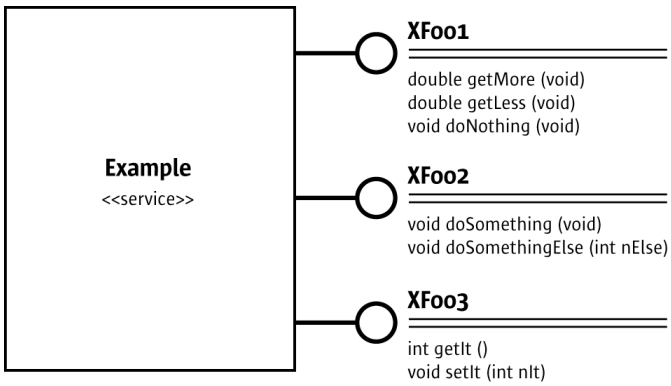


Illustration 27: Basic Hides Interfaces

In Java and C++, it is necessary to obtain a reference to each interface before calling one of its methods. In Basic, every method of every supported interface can be called directly at the object without querying for the appropriate interface in advance. The '.' operator is used:

```
' Basic
oExample = getExampleObjectFromSomewhere()
oExample.doNothing()          ' Calls method doNothing of XFoo1
oExample.doSomething()                ' Calls method doSomething of XFoo2
oExample.doSomethingElse(42)  ' Calls method doSomethingElse of XFoo2
```

Additionally, OpenOffice.org Basic interprets pairs of get and set methods at UNO objects as Basic object properties if they follow this pattern:

```
SomeType getSomeProperty()
void setSomeProperty(SomeType aValue)
```

In this pattern, OpenOffice.org Basic offers a property of type `SomeType` named `SomeProperty`. This functionality is based on the `com.sun.star.beans.Introspection` service. For additional details, see *5.2.3 Advanced UNO - Language Bindings - UNO Reflection API.*

The get and set methods can always be used directly. In our example service above, the methods `getIt()` and `setIt()`, or read and write a Basic property `It` are used:

```
Dim x as Integer
x = oExample.getIt()    ' Calls getIt method of XFoo3

' is the same as

x = oExample.It        ' Read property It represented by XFoo3

oExample.setIt( x )    ' Calls setIt method of XFoo3

' is the same as

oExample.It = x        ' Modify property It represented by XFoo3
```

If there is only a get method, but no associated set method, the property is considered to be read only.

```
Dim x as Integer, y as Integer
x = oExample.getMore()    ' Calls getMore method of XFoo1
y = oExample.getLess()    ' Calls getLess method of XFoo1

' is the same as

x = oExample.More    ' Read property More represented by XFoo1
y = oExample.Less    ' Read property Less represented by XFoo1

' but

oExample.More = x    ' Runtime error "Property is read only"
oExample.Less = y    ' Runtime error "Property is read only"
```

Properties an object provides through `com.sun.star.beans.XPropertySet` are available through the `.` operator. The methods of `com.sun.star.beans.XPropertySet` can be used also. The object `oExample2` in the following example has three `integer` properties `Value1`, `Value2` and `Value3` :

```
Dim x as Integer, y as Integer, z as Integer
x = oExample2.Value1
y = oExample2.Value2
z = oExample2.Value3

' is the same as

x = oExample2.getPropertyValue( "Value1" )
y = oExample2.getPropertyValue( "Value2" )
z = oExample2.getPropertyValue( "Value3" )

' and

oExample2.Value1 = x
oExample2.Value2 = y
oExample2.Value3 = z

' is the same as

oExample2.setPropertyValue( "Value1", x )
oExample2.setPropertyValue( "Value2", y )
oExample2.setPropertyValue( "Value3", z )
```

Basic uses `com.sun.star.container.XNameAccess` to provide named elements in a collection through the `.` operator. However, `XNameAccess` only provides read access. If a collection offers write access through `com.sun.star.container.XNameReplace` or `com.sun.star.container.XNameContainer`, use the appropriate methods explicitly:

```
' oNameAccessible is an object that supports XNameAccess
' including the names "Value1", "Value2"
x = oNameAccessible.Value1
y = oNameAccessible.Value2

' is the same as

x = oNameAccessible.getByName( "Value1" )
y = oNameAccessible.getByName( "Value2" )

' but

oNameAccessible.Value1 = x    ' Runtime Error, Value1 cannot be changed
oNameAccessible.Value2 = y    ' Runtime Error, Value2 cannot be changed

' oNameReplace is an object that supports XNameReplace
' replaceByName() sets the element Value1 to 42
oNameReplace.replaceByName( "Value1", 42 )
```

Instantiating UNO Services

In Basic, instantiate services using the Basic Runtime Library (RTL) function `createUnoService()`. This function expects a fully qualified service name and returns an object supporting this service, if it is available:

```
oSimpleFileAccess = CreateUnoService( "com.sun.star.ucb.SimpleFileAccess" )
```

This call instantiates the `com.sun.star.ucb.SimpleFileAccess` service. To ensure that the function was successful, the returned object can be checked with the `IsNull` function:

```
oSimpleFileAccess = CreateUnoService( "com.sun.star.ucb.SimpleFileAccess" )
bError = IsNull( oSimpleFileAccess )  ' bError is set to False

oNoService = CreateUnoService( "com.sun.star.nowhere.ThisServiceDoesNotExist" )
bError = IsNull( oNoService )         ' bError is set to True
```

Instead of using `CreateUnoService()` to instantiate a service, it is also possible to get the global UNO `com.sun.star.lang.ServiceManager` of the OpenOffice.org process by calling `GetProcessServiceManager()`. Once obtained, use `createInstance()` directly:

```
oServiceMgr = GetProcessServiceManager()
oSimpleFileAccess = oServiceMgr.createInstance( "com.sun.star.ucb.SimpleFileAccess" )

' is the same as

oSimpleFileAccess = CreateUnoService( "com.sun.star.ucb.SimpleFileAccess" )
```

The advantage of `GetProcessServiceManager()` is that additional information and pass in arguments is received when services are instantiated using the service manager. For instance, to initialize a service with arguments, the `createInstanceWithArguments()` method of `com.sun.star.lang.XMultiServiceFactory` has to be used at the service manager, because there is no appropriate Basic RTL function to do that. Example:

```
Dim args(1)
args(0) = "Important information"
args(1) = "Even more important information"
oService = oServiceMgr.createInstanceWithArguments _
    ( "com.sun.star.nowhere.ServiceThatNeedsInitialization", args() )
```

The object returned by `GetProcessServiceManager()` is a normal Basic UNO object supporting `com.sun.star.lang.ServiceManager`. Its properties and methods are accessed as described above.

In addition, the Basic RTL provides special properties as API entry points. They are described in more detail in *11.3 Basic and Dialogs - Features of OpenOffice.org Basic*:

OpenOffice.org Basic RTL Property	Description
ThisComponent	Only exists in Basic code which is embedded in a Writer, Calc, Draw or Impress document. It contains the document model the Basic code is embedded in.
StarDesktop	The `com.sun.star.frame.Desktop` singleton of the office application. It loads document components and handles the document windows. For instance, the document in the top window can be retrieved using `oDoc = StarDesktop.CurrentComponent`

Getting Information about UNO Objects

The Basic RTL retrieves information about UNO objects. There are functions to evaluate objects during runtime and object properties used to inspect objects during debugging.

Checking for interfaces during runtime

Although Basic does not support the `queryInterface` concept like C++ and Java, it can be useful to know if a certain interface is supported by a UNO Basic object or not. The function `HasUnoInterfaces()` detects this.

The first parameter `HasUnoInterfaces()` expects the object that should be tested. Parameter(s) of one or more fully qualified interface names can be passed to the function next. The function returns `True` if all these interfaces are supported by the object, otherwise `False`.

```
Sub Main
    Dim oSimpleFileAccess
    oSimpleFileAccess = CreateUnoService( "com.sun.star.ucb.SimpleFileAccess" )

    Dim bSuccess
    Dim IfaceName1$, IfaceName2$, IfaceName3$
```

```
    IfaceName1$ = "com.sun.star.uno.XInterface"
    IfaceName2$ = "com.sun.star.ucb.XSimpleFileAccess2"
    IfaceName3$ = "com.sun.star.container.XPropertySet"

    bSuccess = HasUnoInterfaces( oSimpleFileAccess, IfaceName1$ )
    MsgBox bSuccess    ' Displays True because XInterface is supported

    bSuccess = HasUnoInterfaces( oSimpleFileAccess, IfaceName1$, IfaceName2$ )
    MsgBox bSuccess    ' Displays True because XInterface
                       ' and XSimpleFileAccess2 are supported

    bSuccess = HasUnoInterfaces( oSimpleFileAccess, IfaceName3$ )
    MsgBox bSuccess    ' Displays False because XPropertySet is NOT supported

    bSuccess = HasUnoInterfaces( oSimpleFileAccess, IfaceName1$, IfaceName2$, IfaceName3$ )
    MsgBox bSuccess    ' Displays False because XPropertySet is NOT supported
End Sub
```

Testing if an object is a struct during runtime

As described in the section *3.4.3 Professional UNO - UNO Language Bindings - OpenOffice.org Basic - Type Mappings - Structs* above, structs are handled differently from objects, because they are treated as values. Use the `IsUnoStruct ()` function to check it the UNO Basic object represents an object or a struct. This function expects one parameter and returns `True` if this parameter is a UNO struct, otherwise `False`. Example:

```
Sub Main
    Dim bIsStruct
        ' Instantiate a service
    Dim oSimpleFileAccess
    oSimpleFileAccess = CreateUnoService( "com.sun.star.ucb.SimpleFileAccess" )
    bIsStruct = IsUnoStruct( oSimpleFileAccess )
    MsgBox bIsStruct    ' Displays False because oSimpleFileAccess is NO struct
                        ' Instantiate a Property struct
    Dim aProperty As New com.sun.star.beans.Property
    bIsStruct = IsUnoStruct( aProperty )
    MsgBox bIsStruct    ' Displays True because aProperty is a struct
    bIsStruct = IsUnoStruct( 42 )
    MsgBox bIsStruct    ' Displays False because 42 is NO struct
End Sub
```

Testing objects for identity during runtime

To find out if two UNO OpenOffice.org Basic objects refer to the same UNO object instance, use the function `EqualUnoObjects()`. Basic is not able to apply the comparison operator = to arguments of type object, for example, `If Obj1 = Obj2 Then` which leads to a runtime error.

Sub Main

```
    Dim bIdentical
    Dim oSimpleFileAccess, oSimpleFileAccess2, oSimpleFileAccess3
        ' Instantiate a service
    oSimpleFileAccess = CreateUnoService( "com.sun.star.ucb.SimpleFileAccess" )
    oSimpleFileAccess2 = oSimpleFileAccess    ' Copy the object reference
    bIdentical = EqualUnoObjects( oSimpleFileAccess, oSimpleFileAccess2 )
    MsgBox bIdentical    ' Displays True because the objects are identical
       ' Instantiate the service a second time
    oSimpleFileAccess3 = CreateUnoService( "com.sun.star.ucb.SimpleFileAccess" )
    bIdentical = EqualUnoObjects( oSimpleFileAccess, oSimpleFileAccess3 )
    MsgBox bIdentical    ' Displays False, oSimpleFileAccess3 is another instance

    bIdentical = EqualUnoObjects( oSimpleFileAccess, 42 )
    MsgBox bIdentical    ' Displays False, 42 is not even an object
       ' Instantiate a Property struct
    Dim aProperty As New com.sun.star.beans.Property
    Dim aProperty2
    aProperty2 = aProperty    ' Copy the struct
    bIdentical = EqualUnoObjects( aProperty, aProperty2 )
    MsgBox bIdentical    ' Displays False because structs are values
                         ' and so aProperty2 is a copy of aProperty
End Sub
```

Basic hides interfaces behind OpenOffice.org Basic objects that could lead to problems when developers are using API structures. It can be difficult to understand the API reference and find the correct method of accessing an object to reach a certain goal.

To assist during development and debugging, every UNO object in OpenOffice.org Basic has special properties that provide information about the object structure. These properties are all

prefixed with `Dbg_` to emphasize their use for development and debugging purposes. The type of these properties is `String`. To display the properties use the `MsgBox` function.

Inspecting interfaces during debugging

The `Dbg_SupportedInterfaces` lists all interfaces supported by the object. In the following example, the object returned by the function `GetProcessServiceManager()` described in the previous section is taken as an example object.

```
oServiceManager = GetProcessServiceManager()
MsgBox oServiceManager.Dbg_SupportedInterfaces
```

This call displays a message box:

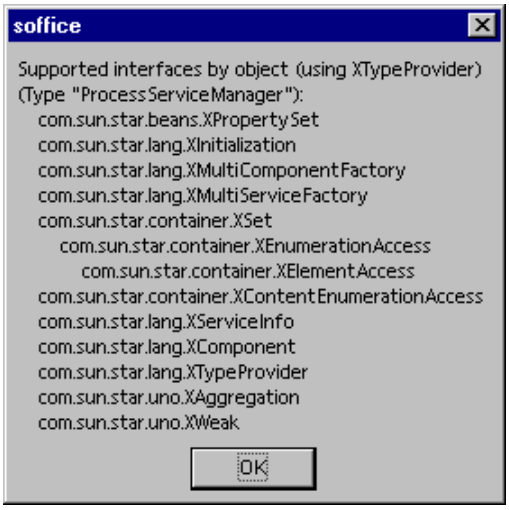


Illustration 28: Dbg_SupportedInterfaces Property

The list contains all interfaces supported by the object. For interfaces that are derived from other interfaces, the super interfaces are indented as shown above for `com.sun.star.container.XSet`, which is derived from `com.sun.star.container.XEnumerationAccess` based upon `com.sun.star.container.XElementAccess`.



If the text "(ERROR: Not really supported!)" is printed behind an interface name, the implementation of the object usually has a bug, because the object pretends to support this interface (per `com.sun.star.lang.XTypeProvider`, but a query for it fails. For details, see *5.2.3 Advanced UNO - Language Bindings - UNO Reflection API*).

Inspecting properties during debugging

The `Dbg_Properties` lists all properties supported by the object through `com.sun.star.beans.XPropertySet` and through get and set methods that could be mapped to Basic object properties:

```
oServiceManager = GetProcessServiceManager()
MsgBox oServiceManager.Dbg_Properties
```

This code produces a message box like this:

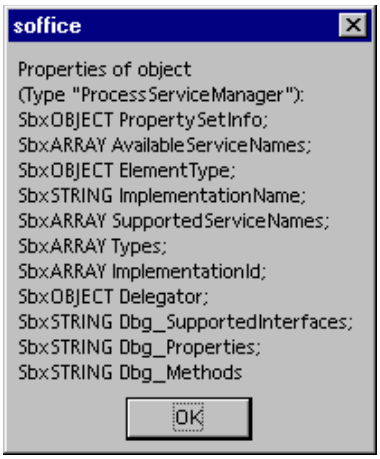


Illustration 29: Dbg_Properties

Inspecting Methods During Debugging

The `Dbg_Methods` lists all methods supported by an object. Example:

```
oServiceManager = GetProcessServiceManager()
MsgBox oServiceManager.Dbg_Methods
```

This code displays:

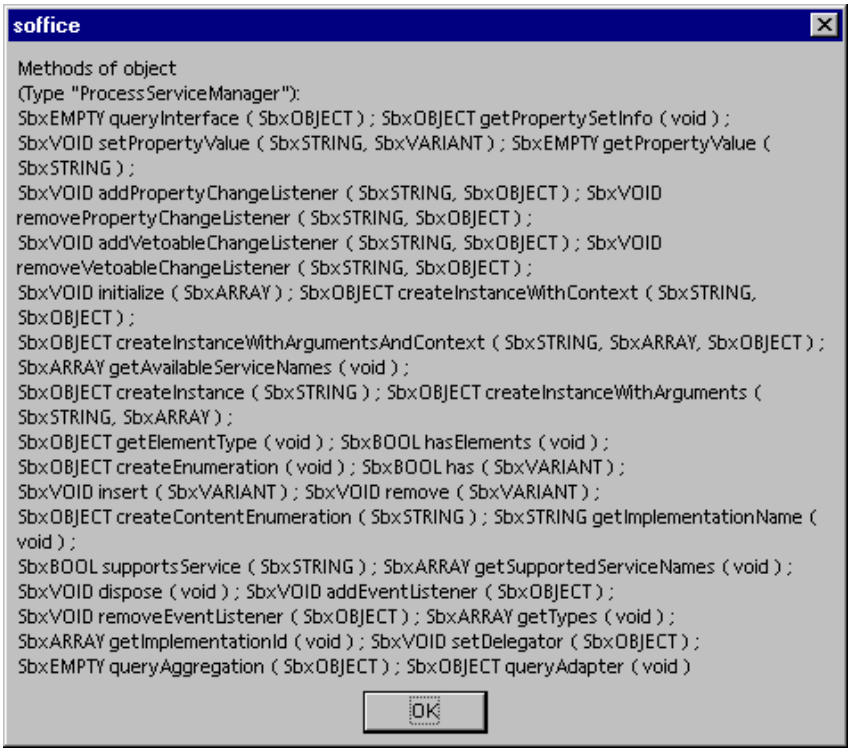


Illustration 30: Dbg_Methods

The notations used in `Dbg_Properties` and `Dbg_Methods` refer to internal implementation type names in Basic. The `Sbx` prefix can be ignored. The remaining names correspond with the normal Basic type notation. The `SbxEMPTY` is the same type as `Variant`. Additional information about Basic types is available in the next chapter.



Basic uses the `com.sun.star.lang.XTypeProvider` interface to detect which interfaces an object supports. Therefore, it is important to support this interface when implementing a component that should be accessible from Basic. For details, see *4 Writing UNO Components*.

Mapping of UNO and Basic Types

Basic and UNO use different type systems. While OpenOffice.orgBasic is compatible to Visual Basic and its type system, UNO types correspond to the IDL specification (see *3.2.1 Professional UNO - API Concepts - Data Types*), therefore it is necessary to map these two type systems. This chapter describes which Basic types have to be used for the different UNO types.

Mapping of Simple Types

In general, the OpenOffice.orgBasic type system is not rigid. Unlike C++ and Java, OpenOffice.orgBasic does not require the declaration of variables, unless the `Option Explicit` command is used that forces the declaration. To declare variables, the `Dim` command is used. Also, a OpenOffice.orgBasic type can be optionally specified through the `Dim` command. The general syntax is:

```
Dim VarName [As Type][, VarName [As Type]]...
```

All variables declared without a specific type have the type `Variant`. Variables of type `Variant` can be assigned values of arbitrary Basic types. Undeclared variables are `Variant` unless type postfixes are used with their names. Postfixes can be used in `Dim` commands as well. The following table contains a complete list of types supported by Basic and their corresponding post-fixes:

Type	Postfix	Range
`Boolean`		`True` or `False`
`Integer`	%	`-32768` to `32767`
`Long`	&	`-2147483648` to `2147483647`
`Single`	!	Floating point number negative: `-3.402823E38` to `-1.401298E-45` positive: `1.401298E-45` to `3.402823E38`
`Double`	#	Double precision floating point number negative: `-1.79769313486232E308` to `-4.94065645841247E-324` positive: `4.94065645841247E-324` to `1.79769313486232E308`
`Currency`	@	Fixed point number with four decimal places `-922,337,203,685,477.5808` to `922,337,203,685,477.5807`
`Date`		`01/01/100` to `12/31/9999`
`Object`		Basic Object
`String`	$	Character string
`Any`		arbitrary Basic type

Consider the following `Dim` examples.

```
Dim a, b            ' Type of a and b is Variant
Dim c as Variant    ' Type of c is Variant

Dim d as Integer    ' Type of d is Integer (16 bit!)

' The type only refers to the preceding variable
Dim e, f as Double  ' ATTENTION! Type of e is Variant!
                    '   Only the type of f is Double
```

```
Dim g as String     ' Type of g is String

Dim i as Date       ' Type of g is Date

' Usage of Postfixes
Dim i%              ' is the same as
Dim i as Integer

Dim d#              ' is the same as
Dim d as Double

Dim s$              ' is the same as
Dim s as String
```

The correlation below is used to map types from UNO to Basic and vice versa.

Uno type	Basic type
long	Long
hyper	Not yet supported
short	Integer
float	Single
double	Double
char	Char (only used internally)
byte	Integer
any	Variant
string	String
boolean	Boolean
void	Void (only used internally)
type	com.sun.star.reflection.XIdlClass

The simple UNO type `type` is mapped to the `com.sun.star.reflection.XIdlClass` interface to retrieve type specific information. For further details, refer to *5.2.3 Advanced UNO - Language Bindings - UNO Reflection API*.

When UNO methods or properties are accessed, and the target UNO type is known, Basic automatically chooses the appropriate types:

```
' The UNO object oExample1 has a property "Count" of type short
a% = 42
oExample1.Count = a%    ' a% has the right type (Integer)

pi = 3,141593
oExample1.Count = pi    ' pi will be converted to short, so Count will become 3

s$ = "111"
oExample1.Count = s$    ' s$ will be converted to short, so Count will become 111
```

Occasionally, OpenOffice.orgBasic does not know the required target type, especially if a parameter of an interface method or a property has the type `any`. In this situation, OpenOffice.orgBasic mechanically converts the OpenOffice.orgBasic type into the UNO type shown in the table above, although a different type may be expected. The only mechanism provided by OpenOffice.orgBasic is an automatic downcast of numeric values:

`Long` and `Integer` values are always converted to the shortest possible integer type:

- to `byte` if `-128 <= Value <= -127`
- to `short` if `-32768 <= Value <= 32767`

The `Single/Double` values are converted to integers in the same manner if they have no decimal places.

This mechanism is used, because some internal C++ tools used to implement UNO functionality in OpenOffice.org provide an automatic upcast but no downcast. Therefore, it can be successful to pass a `byte` value to an interface expecting a `long` value, but not vice versa.

In the following example, `oNameCont` is an object that supports `com.sun.star.container.XNameContainer` and contains elements of type `short`. Assume `FirstValue` is a valid entry.

```
a% = 42
oNameCount.replaceByName( "FirstValue", a% )    ' Ok, a% is downcasted to type byte

b% = 123456
oNameCount.replaceByName( "FirstValue", b% )    ' Fails, b% is outside the short range
```

The method call fails, therefore the implementation should throw the appropriate exception that is converted to a OpenOffice.orgBasic error by the OpenOffice.orgBasic RTL. It may happen that an implementation also accepts unsuitable types and does not throw an exception. Ensure that the values used are suitable for their UNO target by using numeric values that do not exceed the target range or converting them to the correct Basic type before applying them to UNO.

Always use the type `Variant` to declare variables for UNO Basic objects, *not* the type `Object`. The OpenOffice.orgBasic type `Object` is tailored for pure OpenOffice.orgBasic objects and not for UNO OpenOffice.orgBasic objects. The `Variant` variables are best for UNO Basic objects to avoid problems that can result from the OpenOffice.orgBasic specific behavior of the type `Object`:

```
Dim oService1    ' Ok
oService1 = CreateUnoService( "com.sun.star.anywhere.Something" )

Dim oService2 as Object    ' NOT recommended
oService2 = CreateUnoService( "com.sun.star.anywhere.SomethingElse" )
```

Mapping of Sequences and Arrays

Many UNO interfaces use sequences, as well as simple types. The OpenOffice.orgBasic counterpart for sequences are arrays. Arrays are standard elements of the Basic language. The example below shows how they are declared:

```
Dim a1( 100 )    ' Variant array, index range: 0-100 -> 101 elements

Dim a2%( 5 )     ' Integer array, index range: 0-5 -> 6 elements

Dim a3$( 0 )     ' String array, index range: 0-0 -> 1 element

Dim a4&( 9, 19 ) ' Long array, index range: (0-9) x (0-19) -> 200 elements
```

Basic does not have a special index operator like `[]` in C++ and Java. Array elements are accessed using normal parentheses `()`:

```
Dim i%, a%( 10 )
for i% = 0 to 10            ' this loop initializes the array
   a%(i%) = i%
next i%

dim s$
for i% = 0 to 10            ' this loop adds all array elements to a string
   s$ = s$ + " " + a%(i%)
next i%
msgbox s$                   ' Displays the string containing all array elements

Dim b( 2, 3 )
b( 2, 3 ) = 23
b( 0, 0 ) = 0
b( 2, 4 ) = 24              ' Error "Subscript out of range"
```

As the examples show, the indices in `Dim` commands differ from C++ and Java array declarations. They do not describe the number of elements, but the largest allowed index. There is one more array element than the given index. This is important for the mapping of OpenOffice.orgBasic arrays to UNO sequences, because UNO sequences follow the C++/Java array semantic.

When the UNO API requires a sequence, the Basic programmer uses an appropriate array. In the following example, `oSequenceContainer` is an object that has a property `TheSequence` of type `sequence<short>`. To assign a sequence of length 10 with the values 0, 1, 2, ... 9 to this property, the following code can be used:

```
Dim i%, a%( 9 )     ' Maximum index 9 -> 10 elements
for i% = 0 to 9     ' this loop initializes the array
   a%(i%) = i%
next i%

oSequenceContainer.TheSequence = a%()

' If "TheSequence" is based on XPropertySet alternatively
oSequenceContainer.setPropertyValue( "TheSequence", a%() )
```

The Basic programmer must be aware of the different index semantics during programming. In the following example, the programmer passed a sequence with one element, but actually passed two elements:

```
' Pass a sequence of length 1 to the TheSequence property:
Dim a%( 1 )      ' WRONG: The array has 2 elements, not only 1!
a%( 0 ) = 3      ' Only Element 0 is initialized,
                 ' Element 1 remains 0 as initialized by Dim

' Now a sequence with two values (3,0) is passed what
' may result in an error or an unexpected behavior!
oSequenceContainer.setPropertyValue( "TheSequence", a%() )
```



When using Basic arrays as a whole for parameters or for property access, they should always be followed by '`()`' in the Basic code, otherwise errors may occur in some situations.

It can be useful to use a OpenOffice.orgBasic RTL function called `Array()` to create, initialize and assign it to a `Variant` variable in a single step, especially for small sequences:

```
Dim a         ' should be declared as Variant
a = Array( 1, 2, 3 )

' is the same as

Dim a(2)
a( 0 ) = 1
a( 1 ) = 2
a( 2 ) = 3
```

Sometimes it is necessary to pass an empty sequence to a UNO interface. In Basic, empty sequences can be declared by omitting the index from the `Dim` command:

```
Dim a%()     ' empty array/sequence of type Integer

Dim b$()     ' empty array/sequence of String
```

Sequences returned by UNO are also represented in Basic as arrays, but these arrays do not have to be declared as arrays beforehand. Variables used to accept a sequence should be declared as `Variant`. To access an array returned by UNO, it is necessary to get information about the number of elements it contains with the Basic RTL functions `LBound()` and `UBound()`.

The function `LBound()` returns the lower index and `UBound()` returns the upper index. The following code shows a loop going through all elements of a returned sequence:

```
Dim aResultArray    ' should be declared as Variant
aResultArray = oSequenceContainer.TheSequence

dim i%, iFrom%, iTo%
iFrom% = LBound( aResultArray() )
iTo%   = UBound( aResultArray() )
for i% = iFrom% to iTo%     ' this loop displays all array elements
   msgbox aResultArray(i%)
next i%
```

The function `LBound()` is a standard Basic function and is not specific in a UNO context. Basic arrays do not necessarily start with index 0, because it is possible to write something similar to:

```
Dim a (3 to 5 )
```

This causes the array to have a lower index of 3. However, sequences returned by UNO always have the start index 0. Usually only `UBound()` is used and the example above can be simplified to:

```
Dim aResultArray    ' should be declared as Variant
aResultArray = oSequenceContainer.TheSequence

Dim i%, iTo%
iTo%   = UBound( aResultArray() )
For i% = 0 To iTo%    ' this loop displays all array elements
   MsgBox aResultArray(i%)
Next i%
```

The element count of a sequence/array can be calculated easily:

```
u% = UBound( aResultArray() )
ElementCount% = u% + 1
```

For empty arrays/sequences `UBound` returns `-1`. This way the semantic of `UBound` stays consistent as the element count is then calculated correctly as:

```
ElementCount% = u% + 1        ' = -1 + 1 = 0
```



The mapping between UNO sequences and Basic arrays depends on the fact that both use a zero-based index system. To avoid problems, the syntax
`Dim a ( IndexMin to IndexMin )`
or the Basic command `Option Base 1` should not be used. Both cause all Basic arrays to start with an index other than 0.

UNO also supports sequences of sequences. In Basic, this corresponds with arrays of arrays. Do not mix up sequences of sequences with multidimensional arrays. In multidimensional arrays, all sub arrays always have the same number of elements, whereas in sequences of sequences every element sequence can have a different size. Example:

```
Dim aArrayOfArrays    ' should be declared as Variant
aArrayOfArrays = oExample.ShortSequences    ' returns a sequence of sequences of short

Dim i%, NumberOfSequences%
Dim j%, NumberOfElements%
Dim aElementArray

NumberOfSequences% = UBound( aArrayOfArrays() ) + 1
For i% = 0 to NumberOfSequences% - 1  ' loop over all sequences
   aElementArray = aArrayOfArrays( i% )
   NumberOfElements% = UBound( aElementArray() ) + 1

   For j% = 0 to NumberOfElements% - 1    ' loop over all elements
      MsgBox aElementArray( j% )
   Next j%
Next i%
```

To create an array of arrays in Basic, sub arrays are used as elements of a master array:

```
' Declare master array
Dim aArrayOfArrays( 2 )

' Declare sub arrays
Dim aArray0( 3 )
Dim aArray1( 2 )
Dim aArray2( 0 )

' Initialise sub arrays
aArray0( 0 ) = 0
aArray0( 1 ) = 1
aArray0( 2 ) = 2
aArray0( 3 ) = 3

aArray1( 0 ) = 42
aArray1( 1 ) = 0
aArray1( 2 ) = -42

aArray2( 0 ) = 1

' Assign sub arrays to the master array
aArrayOfArrays( 0 ) = aArray0()
aArrayOfArrays( 1 ) = aArray1()
aArrayOfArrays( 2 ) = aArray2()

' Assign the master array to the array property
oExample.ShortSequences = aArrayOfArrays()
```

In this situation, the runtime function `Array()` is useful. The example code can then be written much shorter:

```
' Declare master array
Dim aArrayOfArrays( 2 )

' Create and assign sub arrays
aArrayOfArrays( 0 ) = Array( 0, 1, 2, 3 )
aArrayOfArrays( 1 ) = Array( 42, 0, -42 )
aArrayOfArrays( 2 ) = Array( 1 )

' Assign the master array to the array property
oExample.ShortSequences = aArrayOfArrays()
```

If you nest `Array()`, more compact code can be written, but it becomes difficult to understand the resulting arrays:

```
' Declare master array variable as variant
Dim aArrayOfArrays

' Create and assign master array and sub arrays
aArrayOfArrays = Array( Array( 0, 1, 2, 3 ), Array( 42, 0, -42 ), Array( 1 ) )

' Assign the master array to the array property
oExample.ShortSequences = aArrayOfArrays()
```

Sequences of higher order can be handled accordingly.

Mapping of Structs

UNO struct types can be instantiated with the `Dim As New` command as a single instance and array.

```
' Instantiate a Property struct
Dim aProperty As New com.sun.star.beans.Property

' Instantiate an array of Locale structs
Dim Locales(10) As New com.sun.star.lang.Locale
```

UNO struct instances are handled like UNO objects. Struct members are accessed using the . operator. The `Dbg_Properties` property is supported. The properties `Dbg_SupportedInterfaces` and `Dbg_Methods` are not supported because they do not apply to structs.:

```
' Instantiate a Locale struct
Dim aLocale As New com.sun.star.lang.Locale

' Display properties
MsgBox aLocale.Dbg_Properties

' Access "Language" property
aLocale.Language = "en"
```

Objects and structs are different. Objects are handled as references and structs as values. When structs are assigned to variables, the structs are copied. This is important when modifying an object property that is a struct, because a struct property has to be reassigned to the object after reading and modifying it.

In the following example, `oExample` is an object that has the properties `MyObject` and `MyStruct`.

- The object provided by `MyObject` supports a string property `ObjectName`.
- The struct provided by `MyStruct` supports a string property `StructName`.

Both `oExample.MyObject.ObjectName` and `oExample.MyStruct.StructName` should be modified. The following code shows how this is done for an object:

```
' Accessing the object
Dim oObject
oObject = oExample.MyObject
oObject.ObjectName = "Tim"    ' Ok!

    ' or shorter

oExample.MyObject.ObjectName = "Tim"    ' Ok!
```

The following code shows how it is done correctly for the struct (and possible mistakes):

```
' Accessing the struct
Dim aStruct
aStruct = oExample.MyStruct    ' aStruct is a copy of oExample.MyStruct!
aStruct.StructName = "Tim"     ' Affects only the property of the copy!

    ' If the code ended here, oExample.MyStruct wouldn't be modified!

oExample.MyStruct = aStruct    ' Copy back the complete struct! Now it's ok!


' Here the other variant does NOT work at all, because
' only a temporary copy of the struct is modified!
oExample.MyStruct.StructName = "Tim"    ' WRONG! oExample.MyStruct is not modified!
```

Mapping of Enums and Constant Groups

Use the fully qualified names to address the values of an enum type by their names. The following examples assume that `oExample` and `oExample2` support `com.sun.star.beans.XPropertySet` with a property `Status` of the enum type `com.sun.star.beans.PropertyState`:

```
Dim EnumValue
EnumValue = com.sun.star.beans.PropertyState.DEFAULT_VALUE
MsgBox EnumValue    ' displays 1

eExample.Status = com.sun.star.beans.PropertyState.DEFAULT_VALUE
```

Basic does not support Enum types. In Basic, enum values coming from UNO are converted to `Long` values. As long as Basic knows if a property or an interface method parameter expects an enum type, then the `Long` value is internally converted to the right enum type. Problems appear with Basic when interface access methods expect an `Any`:

```
Dim EnumValue
EnumValue = oExample.Status    ' EnumValue is of type Long

' Accessing the property implicitly
oExample2.Status = EnumValue  ' Ok! EnumValue is converted to the right enum type

' Accessing the property explicitly using XPropertySet methods
oExample2.setPropertyValue( "Status", EnumValue )    ' WRONG! Will probably fail!
```

The explicit access could fail, because `EnumValue` is passed as parameter of type `Any` to `setPropertyValue()`, therefore Basic does not know that a value of type PropertyState is expected. There is still a problem, because the Basic type for `com.sun.star.beans.PropertyState` is `Long`. This problem is solved in the implementation of the `com.sun.star.beans.XPropertySet` interface. For enum types, the implicit property access using the Basic property syntax `Object.Property` is preferred to calling generic methods using the type `Any`. In situations where only a generic interface method that expects an `enum` for an `Any`, there is no solution for Basic.

Constant groups are used to specify a set of constant values in IDL. In Basic, these constants can be accessed using their fully qualified names. The following code displays some constants from `com.sun.star.beans.PropertyConcept`:

```
MsgBox com.sun.star.beans.PropertyConcept.DANGEROUS    ' Displays 1
MsgBox com.sun.star.beans.PropertyConcept.PROPERTYSET  ' Displays 2
```

A constant group or enum can be assigned to an object. This method is used to shorten code if more than one enum or constant value has to be accessed:

```
Dim oPropConcept
oPropConcept = com.sun.star.beans.PropertyConcept
msgbox oPropConcept.DANGEROUS    ' Displays 1
msgbox oPropConcept.PROPERTYSET  ' Displays 2
```

Case Sensitivity

Generally Basic is case insensitive. However, this does not always apply to the communication between UNO and Basic. To avoid problems with case sensitivity write the UNO related code as if

Basic was case sensitive. This facilitates the translation of a Basic program to another language, and Basic code becomes easier to read and understand. The following discusses problems that might occur.

Identifiers that differ in case are considered to be identical when they are used with UNO object properties, methods and struct members.

```
Dim ALocale As New com.sun.star.lang.Locale
alocale.language = "en"     ' Ok
MsgBox aLocale.Language     ' Ok
```

The exceptions to this is if a Basic property is obtained through `com.sun.star.container.XNameAccess` as described above, its name has to be written exactly as it is in the API reference. Basic uses the name as a string parameter that is not interpreted when accessing `com.sun.star.container.XNameAccess` using its methods.

' oNameAccessible is an object that supports XNameAccess

```
' including the names "Value1", "Value2"
x = oNameAccessible.Value1    ' Ok
y = oNameAccessible.VaLUe2    ' Runtime Error, Value2 is not written correctly

    ' is the same as

x = oNameAccessible.getByName( "Value1" )     ' Ok
y = oNameAccessible.getByName( "VaLUe2" )     ' Runtime Error, Value2 is not written correctly
```

Exception Handling

Unlike UNO, Basic does not support exceptions. All exceptions thrown by UNO are caught by the Basic runtime system and transformed to a Basic error. Executing the following code results in a Basic error that interrupts the code execution and displays an error message:

```
Sub Main
    Dim oLib
    oLib = BasicLibraries.getByName( "InvalidLibraryName" )
End Sub
```

The `BasicLibraries` object used in the example contains all the available Basic libraries in a running office instance. The Basic libraries contained in `BasicLibraries` is accessed using `com.sun.star.container.XNameAccess`. An exception was provoked by trying to obtain a non-existing library. The BasicLibraries object is explained in more detail in *11.4 Basic and Dialogs - Advanced Library Organization.*

The call to `getByName()` results in this error box:



Illustration 31: Unhandled UNO Exception

However, the Basic runtime system is not always able to recognize the Exception type. Sometimes only the exception message can be displayed that has to be provided by the object implementation.

Exceptions transformed to Basic errors can be handled just like any Basic error using the `On Error GoTo` command:

```
Sub Main
    On Error Goto ErrorHandler    ' Enables error handling
```

```
        Dim oLib
        oLib = BasicLibraries.getByName( "InvalidLibraryName" )
        MsgBox "After the Error"
        Exit Sub

    ' Label
    ErrorHandler:
        MsgBox "Error code: " + Err + Chr$(13) + Error$
        Resume Next    ' Continues execution at the command following the error command
    End Sub
```

When the exception occurs, the execution continues at the `ErrorHandler` label. In the error handler, some properties are used to get information about the error. The `Err` is the error code that is 1 for UNO exceptions. The `Error$` contains the text of the error message. Executing the program results in the following output:



Illustration 32: Handled UNO Exception

Another message box "After the Error" is displayed after the above dialog box, because `Resume Next` goes to the code line below the line where the exception was thrown. The `Exit Sub` command is required so that the error handler code would be executed again.

Listeners

Many interfaces in UNO are used to register listener objects implementing special listener interfaces, so that a listener gets feedback when its appropriate listener methods are called. OpenOffice.org Basic does not support the concept of object implementation, therefore a special RTL function named `CreateUnoListener()` has been introduced. It uses a prefix for method names that can be called back from UNO. The `CreateUnoListener()` expects a method name prefix and the type name of the desired listener interface. It returns an object that supports this interface that can be used to register the listener.

The following example instantiates an `com.sun.star.container.XContainerListener`. Note the prefix `ContListener_`:

```
    Dim oListener
    oListener = CreateUnoListener( "ContListener_", "com.sun.star.container.XContainerListener" )
```

The next step is to implement the listener methods. In this example, the listener interface has the following methods:

Methods of `com.sun.star.container.XContainerListener`	
`disposing()`	Method of the listener base interface `com.sun.star.lang.XEventListener`, contained in every listener interface, because all listener interfaces must be derived from this base interface. Takes a `com.sun.star.lang.EventObject`
`elementInserted()`	Method of interface `com.sun.star.container.XContainerListener`. Takes a `com.sun.star.container.ContainerEvent`.
`elementRemoved()`	Method of interface `com.sun.star.container.XContainerListener`. Takes a `com.sun.star.container.ContainerEvent`.
`elementReplaced()`	Method of interface `com.sun.star.container.XContainerListener`. Takes a `com.sun.star.container.ContainerEvent`.

In the example, `ContListener_` is specified as a name prefix, therefore the following subs have to be implemented in Basic.

- `ContListener_disposing`
- `ContListener_elementInserted`
- `ContListener_elementRemoved`
- `ContListener_elementReplaced`

Every listener type has a corresponding `Event` struct type that contains information about the event. When a listener method is called, an instance of this `Event` type is passed as a parameter. In the Basic listener methods these `Event` objects can be evaluated by adding an appropriate `Variant` parameter to the procedure header. The following code shows how the listener methods in the example could be implemented:

```
Sub ContListener_disposing( oEvent )
    MsgBox "disposing"
    MsgBox oEvent.Dbg_Properties
End Sub

Sub ContListener_elementInserted( oEvent )
    MsgBox "elementInserted"
    MsgBox oEvent.Dbg_Properties
End Sub

Sub ContListener_elementRemoved( oEvent )
    MsgBox "elementRemoved"
    MsgBox oEvent.Dbg_Properties
End Sub

Sub ContListener_elementReplaced( oEvent )
    MsgBox "elementReplaced"
    MsgBox oEvent.Dbg_Properties
End Sub
```

It is necessary to implement *all* listener methods, including the listener methods of the parent interfaces of a listener. Basic runtime errors will occur whenever an event occurs and no corresponding Basic sub is found, especially with `disposing()`, because the broadcaster might be destroyed a long time after the Basic program was ran. In this situation, Basic shows a "Method not found" message. There is no indication of which method cannot be found or why Basic is looking for a method.

We are listening for events at the basic library container. Our simple implementation for events triggered by user actions in the **Tools - Macro - Organizer** dialog displays a message box with the corresponding listener method name and a message box with the `Dbg_Properties` of the event struct. For the `disposing()` method, the type of the event object is `com.sun.star.lang.EventObject`. All other methods belong to `com.sun.star.container.XContainerListener`, therefore the type of the event object is `com.sun.star.container.ContainerEvent`. This type is derived from `com.sun.star.lang.EventObject` and contains additional container related information.

If the event object is not needed, the parameter could be left out of the implementation. For example, the `disposing()` method could be:

```
' Minimal implementation of Sub disposing
Sub ContListener_disposing
End Sub
```

The event objects passed to the listener methods can be accessed like other struct objects. The following code shows an enhanced implementation of the `elementRemoved()` method that evaluates the `com.sun.star.container.ContainerEvent` to display the name of the module removed from `Library1` and the module source code:

```
sub ContListener_ElementRemoved( oEvent )
    MsgBox "Element " + oEvent.Accessor + " removed"
    MsgBox "Source =" + Chr$(13) + Chr$(13) + oEvent.Element
End Sub
```

When the user removes `Module1`, the following message boxes are displayed by `ContListener_ElementRemoved()`:

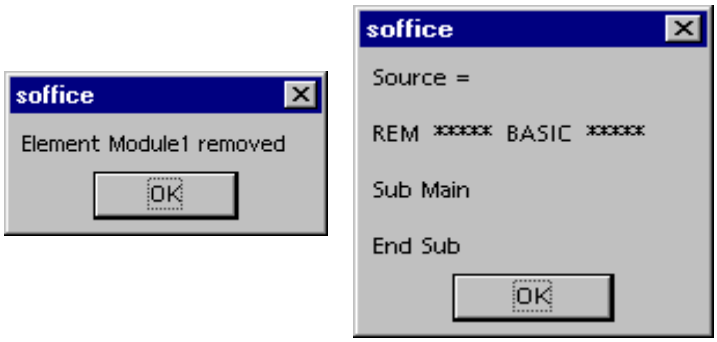


Illustration 33: ContListener_ElementRemoved Event Callback

When all necessary listener methods are implemented, add the listener to the broadcaster object by calling the appropriate add method. To register an `XContainerListener`, the corresponding registration method at our container is `addContainerListener()`:

```
Dim oLib
oLib = BasicLibraries.Library1            ' Library1 must exist!
oLib.addContainerListener( oListener )    ' Register the listener
```



The naming scheme `XSomeEventListener <> addSomeEventListener()` is used throughout the OpenOffice.org API.

The listener for container events is now registered permanently. When a container event occurs, the container calls the appropriate method of the `com.sun.star.container.XContainerListener` interface in our Basic code.

3.4.4 Automation Bridge

Introduction

The OpenOffice.org software supports Microsoft's *Automation* technology. This offers programmers the possibility to control the office from external programs. There is a range of efficient IDEs and tools available for developers to choose from.

Automation is language independent. The respective compilers or interpreters must, however, support Automation. The compilers transform the source code into Automation compatible computing instructions. For example, the string and array types of your language can be used without caring about their internal representation in Automation, which is `BSTR` and `SAFEARRAY`. A client program that controls OpenOffice.org can be represented by an executable (Visual Basic, C++) or a script (JScript, VB Script). The latter requires an additional program to run the scripts, such as Windows Scripting Host (WSH) or Internet Explorer.

UNO was not designed to be compatible with Automation and COM, although there are similarities. OpenOffice.org deploys a bridging mechanism provided by the *Automation Bridge* to make UNO and Automation work together. The bridge consists of UNO services, however, it is not necessary to have a special knowledge about them to write Automation clients for OpenOffice.org. For additional information, refer to (see *3.4.4 Professional UNO - UNO Language Bindings - Automation Bridge - The Bridge Services*).

Different languages have different capabilities. There are differences in the manner that the same task is handled, depending on the language used. Examples in Visual Basic, VB Script and JScript are provided. They will show when a language requires special handling or has a quality to be aware of. Although Automation is supposed to work across languages, there are subtleties that require a particular treatment by the bridge or a style of coding. For example, JScript does not know `out` parameters, therefore `Array` objects have to be used. Currently, the bridge has been tested with C++, JScript, VBScript and Visual Basic, although other languages can be used as well.

The name *Automation Bridge* implies the use of the Automation technology. Automation is part of the collection of technologies commonly referred to as ActiveX or OLE, therefore the term OLE Bridge is misleading and should be avoided. Also, the bridge only supports the COM interfaces `IDispatch` and `IdispatchEX`. Refer to *3.4.4 Professional UNO - UNO Language Bindings - Automation Bridge - Unsupported COM Features* for the extent these interfaces are supported.

Requirements

The Automation technology can only be used with OpenOffice.org on a Windows platform (Windows 95, 98, NT4, ME, 2000, XP). There are COM implementations on Macintosh OS and UNIX, but there has been no effort to support Automation on these platforms.

Using Automation involves creating objects in a COM-like fashion, that is, using functions like `CreateObject()` in VB or `CoCreateInstance()` in C. This requires the OpenOffice.org automation objects to be registered with the Windows system registry. This registration is carried out whenever an office is installed on the system. If the registration did not take place, for example because the binaries were just copied to a certain location, then Automation clients will not work correctly or not at all. Refer to *3.4.4 Professional UNO - UNO Language Bindings - Automation Bridge - The Service Manager Component* for additional information.

A Quick Tour

The following example shows how to access OpenOffice.org functionality through Automation. Note the inline comments. The only automation specific call is `WScript.CreateObject()` in the first line, the remaining are OpenOffice.org API calls. The helper functions `createStruct()` and `insertIntoCell()` are shown at the end of the listing

```
'This is a VBScript example
'The service manager is always the starting point
'If there is no office running then an office is started up
Set objServiceManager= WScript.CreateObject("com.sun.star.ServiceManager")

'Create the CoreReflection service that is later used to create structs
Set objCoreReflection= objServiceManager.createInstance("com.sun.star.reflection.CoreReflection")

'Create the Desktop
Set objDesktop= objServiceManager.createInstance("com.sun.star.frame.Desktop")

'Open a new empty writer document
Dim args()
Set objDocument= objDesktop.loadComponentFromURL("private:factory/swriter", "_blank", 0, args)

'Create a text object
Set objText= objDocument.getText

'Create a cursor object
Set objCursor= objText.createTextCursor

'Inserting some Text
objText.insertString objCursor, "The first line in the newly created text document." & vbLf, false

'Inserting a second line
objText.insertString objCursor, "Now we're in the second line", false

'Create instance of a text table with 4 columns and 4 rows
Set objTable= objDocument.createInstance( "com.sun.star.text.TextTable")
```

```
objTable.initialize 4, 4

'Insert the table
objText.insertTextContent objCursor, objTable, false

'Get first row
Set objRows= objTable.getRows
Set objRow= objRows.getByIndex( 0)

'Set the table background color
objTable.setPropertyValue "BackTransparent", false
objTable.setPropertyValue "BackColor", 13421823

'Set a different background color for the first row
objRow.setPropertyValue "BackTransparent", false
objRow.setPropertyValue "BackColor", 6710932

'Fill the first table row
insertIntoCell "A1","FirstColumn", objTable // insertIntoCell is a helper function, see below
insertIntoCell "B1","SecondColumn", objTable
insertIntoCell "C1","ThirdColumn", objTable
insertIntoCell "D1","SUM", objTable

objTable.getCellByName("A2").setValue 22.5
objTable.getCellByName("B2").setValue 5615.3
objTable.getCellByName("C2").setValue -2315.7
objTable.getCellByName("D2").setFormula"sum "

objTable.getCellByName("A3").setValue 21.5
objTable.getCellByName("B3").setValue 615.3
objTable.getCellByName("C3").setValue -315.7
objTable.getCellByName("D3").setFormula "sum "

objTable.getCellByName("A4").setValue 121.5
objTable.getCellByName("B4").setValue -615.3
objTable.getCellByName("C4").setValue 415.7
objTable.getCellByName("D4").setFormula "sum "

'Change the CharColor and add a Shadow
objCursor.setPropertyValue "CharColor", 255
objCursor.setPropertyValue "CharShadowed", true

'Create a paragraph break
'The second argument is a com::sun::star::text::ControlCharacter::PARAGRAPH_BREAK constant
objText.insertControlCharacter objCursor, 0 , false

'Inserting colored Text.
objText.insertString objCursor, " This is a colored Text - blue with shadow" & vbLf, false

'Create a paragraph break ( ControlCharacter::PARAGRAPH_BREAK).
objText.insertControlCharacter objCursor, 0, false

'Create a TextFrame.
Set objTextFrame= objDocument.createInstance("com.sun.star.text.TextFrame")

'Create a Size struct.
Set objSize= createStruct("com.sun.star.awt.Size") // helper function, see below
objSize.Width= 15000
objSize.Height= 400
objTextFrame.setSize( objSize)

' TextContentAnchorType.AS_CHARACTER = 1
objTextFrame.setPropertyValue "AnchorType", 1

'insert the frame
objText.insertTextContent objCursor, objTextFrame, false

'Get the text object of the frame
Set objFrameText= objTextFrame.getText

'Create a cursor object
Set objFrameTextCursor= objFrameText.createTextCursor

'Inserting some Text
objFrameText.insertString objFrameTextCursor, "The first line in the newly created text frame.", _
false
objFrameText.insertString objFrameTextCursor, _
vbLf & "With this second line the height of the frame raises.", false

'Create a paragraph break
'The second argument is a com::sun::star::text::ControlCharacter::PARAGRAPH_BREAK constant
objFrameText.insertControlCharacter objCursor, 0 , false

'Change the CharColor and add a Shadow
objCursor.setPropertyValue "CharColor", 65536
objCursor.setPropertyValue "CharShadowed", false
```

```
'Insert another string
objText.insertString objCursor, " That's all for now !!", false

On Error Resume Next
    If Err Then
    MsgBox "An error occurred"
End If


Sub insertIntoCell( strCellName, strText, objTable)
    Set objCellText= objTable.getCellByName( strCellName)
    Set objCellCursor= objCellText.createTextCursor
    objCellCursor.setPropertyValue "CharColor",16777215
    objCellText.insertString objCellCursor, strText, false
End Sub

Function createStruct( strTypeName)
    Set classSize= objCoreReflection.forName( strTypeName)
    Dim aStruct
    classSize.createObject aStruct
    Set createStruct= aStruct
End Function
```

This script created a new document and started the office, if necessary. The script also wrote text, created and populated a table, used different background and pen colors. Only one object is created as an ActiveX component called `com.sun.star.ServiceManager`. The service manager is then used to create additional objects which in turn provided other objects. All those objects provide functionality that can be used by invoking the appropriate functions and properties. A developer must learn which objects provide the desired functionality and how to obtain them. The chapter *2 First Steps* introduces the main OpenOffice.org objects available to the programmer.

The Service Manager Component

Instantiation

The service manager is the starting point for all Automation clients. The service manager requires to be created before obtaining any UNO object. Since the service manager is a COM component, it has a `CLSID` and a programmatic identifier which is `com.sun.star.ServiceManager`. It is instantiated like any ActiveX component, depending on the language used:

```
//C++
IDispatch* pdispFactory= NULL;
CLSID clsFactory= {0x82154420,0x0FBF,0x11d4,{0x83, 0x13,0x00,0x50,0x04,0x52,0x6A,0xB4}};
hr= CoCreateInstance( clsFactory, NULL, CLSCTX_ALL, __uuidof(IDispatch), (void**)&pdispFactory);
```

In Visual C++, use classes which facilitate the usage of COM pointers. If you use the Active Template Library (ATL), then the following example looks like this:

```
CComPtr<IDispatch> spDisp;
if( SUCCEEDED( spDisp.CoCreateInstance("com.sun.star.ServiceManager")))
{
    // do something
}
```

JScript:

```
var objServiceManager= new ActiveXObject("com.sun.star.ServiceManager");
```

Visual Basic:

```
Dim objManager As Object
Set objManager= CreateObject("com.sun.star.ServiceManager")
```

VBScript with WSH:

```
Set objServiceManager= WScript.CreateObject("com.sun.star.ServiceManager")
```

JScript with WSH:

```
var objServiceManager= WScript.CreateObject("com.sun.star.ServiceManager");
```

The service manager can also be created remotely, that is. on a different machine, taking the security aspects into account. For example, set up launch and access rights for the service manager in the system registry (see “DCOM”).

The code for the service manager resides in the office executable *soffice.exe*. COM starts up the executible whenever a client tries to obtain the class factory for the service manager, so that the client can use it.

Registry Entries

For the instantiation to succeed, the service manager must be properly registered with the system registry. The keys and values shown in the tables below are all written during setup. It is not necessary to edit them to use the Automation capability of the office. Automation works immediately after installation. There are three different keys under `HKEY_CLASSES_ROOT` that have the following values and subkeys:

Key	**Value**
`CLSID\{82154420-0FBF-11d4-8313-005004526AB4}`	`"StarOffice Service Manager (Ver 1.0)"`
Sub Keys	
`LocalServer32`	`"<OfficePath>\program\soffice.exe”`
`NotInsertable`	
`ProgIDcom.sun.star.ServiceManager.1`	`"com.sun.star.ServiceManager.1"`
`Programmable`	
`VersionIndependentProgID`	`"com.sun.star.ServiceManager"`

Key	**Value**
`com.sun.star.ServiceManager`	`"StarOffice Service Manager"`
Sub Keys	
`CLSID`	`"{82154420-0FBF-11d4-8313-005004526AB4}"`
`CurVer`	`"com.sun.star.ServiceManager.1"`

Key	**Value**
`com.sun.star.ServiceManager.1`	`"StarOffice Service Manager (Ver 1.0)"`
Sub Keys	
`CLSID`	`"{82154420-0FBF-11d4-8313-005004526AB4}"`

The value of the key `CLSID\{82154420-0FBF-11d4-8313-005004526AB4}\LocalServer32` reflects the path of the office executable.

All keys have duplicates under `HKEY_LOCAL_MACHINE\SOFTWARE\Classes\`.

The service manager is an ActiveX component, but does not support self-registration. That is, the office does not support the command line arguments `-RegServer` or `-UnregServer`.
The service manager, as well as all the objects that it creates and that originate from it indirectly as return values of function calls are proper automation objects. They can also be accessed remotely through DCOM.

From UNO Objects to Automation Objects

The service manager is based on the UNO service manager and similar to all other UNO components, is not compatible with Automation. The service manager can be accessed through the COM API, because the service manager is an Active X component contained in an executable that is the OpenOffice.org. When a client creates the service manager, for example by calling `CreateObject()`, and the office is not running, it is started up by the COM system. The office then creates a class factory for the service manager and registers it with COM. At that point, COM uses the factory to instantiate the service manager and return it to the client.

When the function `IClassFactory::CreateInstance` is called, the UNO service manager is converted into an Automation object. The actual conversion is carried out by the UNO service `com.sun.star.bridge.OleBridgeSupplier2` (see *3.4.4 Professional UNO - UNO Language Bindings - Automation Bridge - The Bridge Services*). The resulting Automation object contains the UNO object and translates calls to `IDispatch::Invoke` into calls to the respective UNO interface function. The supplied function arguments, as well as the return values of the UNO function are converted according to the defined mappings (see *3.4.4 Professional UNO - UNO Language Bindings - Automation Bridge - Type Mappings*). Returned objects are converted into Automation objects, so that all objects obtained are always proper Automation objects.

Using UNO from Automation

With the IDL descriptions and documentation, start writing code that uses an interface. This requires knowledge about the programming language, especially how UNO interfaces can be accessed in that language and how function calls work.

In some languages, such as C++, the use of interfaces and their functions is simple, because the IDL descriptions map well with the respective C++ counterparts. For example, the syntax of functions are similar, and interfaces and out parameters can also be realized. The C++ language is not the best choice for Automation, because all interface calls have to use `IDispatch`, which is difficult to use in C++. In other languages, such as VB and Jscript, the `IDispatch` interface is hidden behind an object syntax that leads to shorter and more understandable code.

Different interfaces can have functions with the same name. There is no way to call a function which belongs to a particular interface, because interfaces can not be requested in Automation . If a UNO object provides two functions with the same name, it is undefined which function will be called. A solution for this issue is planned for the future.

Not all languages treat method parameters in the same manner, especially when it comes to input parameters that are reused as output parameters. From the perspective of a VB programmer an out parameter does not look different from an in parameter. However, to realize out parameters in Jscript, use an `Array` or `Value Object` that is a special construct provided by the Automation bridge. JScript does not support out parameters through calls by reference.

Calling Functions and Accessing Properties

The essence of Automation objects is the `IDispatch` interface. All function calls, including the access to properties, ultimately require a call to `IDispatch::Invoke`. When using C++, the use of `IDispatch` is rather cumbersome. For example, the following code calls `createInstance("com.sun.star.reflection.CoreReflection")`:

OLECHAR* **`funcName= L"createInstance"`**;

```
DISPID id;
IDispatch* pdispFactory= NULL;
CLSID clsFactory= {0x82154420,0x0FBF,0x11d4,{0x83, 0x13,0x00,0x50,0x04,0x52,0x6A,0xB4}};
HRESULT hr= CoCreateInstance( clsFactory, NULL, CLSCTX_ALL, __uuidof(IDispatch), (void**)&pdispFactory);
```

```
if( SUCCEEDED(pdispFactory->GetIDsOfNames( IID_NULL, &funcName, 1, LOCALE_USER_DEFAULT, &id)))
{
    VARIANT param1;
    VariantInit( &param1);
    param1.vt= VT_BSTR;
    param1.bstrVal=   SysAllocString( L"com.sun.star.reflection.CoreReflection");
    DISPPARAMS dispparams= { &param1, 0, 1, 0};
    VARIANT result;
    VariantInit( &result);
    hr= pdispFactory->Invoke( id, IID_NULL, LOCALE_USER_DEFAULT, DISPATCH_METHOD,
                        &dispparams, &result, NULL, 0);
}
```

First the COM ID for the method name `createInstance()` is retrieved from `GetIdsOfNames`, then the ID is used to `invoke()` the method `createInstance()`.

Before calling a certain function on the `IDispatch` interface, get the `DISPID` by calling `GetIDsOfNames`. The `DISPID`s are generated by the bridge, as required. There is no fixed mapping from member names to `DISPID`s, that is, the `DISPID` for the same function of a second instance of an object might be different. Once a `DISPID` is created for a function or property name, it remains the same during the lifetime of this object.

Helper classes can make it easier. The next example shows the same call realized with helper classes from the Active Template Library:

CComDispatchDriver spDisp(pdispFactory);

```
CComVariant param(L"com.sun.star.reflection.CoreReflection");
CComVariant result;
hr= spUnk.Invoke1(L"createInstance",param, result);
```

Some frameworks allow the inclusion of COM type libraries that is an easier interface to Automation objects during development. These helpers cannot be used with UNO, because the SDK does not provide COM type libraries for UNO components. While COM offers various methods to invoke functions on COM objects, UNO supports IDispatch only.

Programming of Automation objects is simpler with VB or JScript, because the `IDispatch` interface is hidden and functions can be called directly. Also, there is no need to wrap the arguments into `VARIANT`s.

```
//VB
Dim objRefl As Object
Set objRefl= dispFactory.createInstance("com.sun.star.reflection.CoreReflection")

//JScript
var objRefl= dispFactory.createInstance("com.sun.star.reflection.CoreReflection");
```

Pairs of get/set functions following the pattern

```
SomeType getSomeProperty()
void setSomeProperty(SomeType aValue)
```

are handled as COM object properties.

Accessing such a property in C++ is similar to calling a method. First, obtain a `DISPID`, then call `IDispatch::Invoke` with the proper arguments.

```
    DISPID dwDispID;
    VARIANT value;
    VariantInit(&value);
    OLECHAR* name= L"AttrByte";
    HRESULT hr = pDisp->GetIDsOfNames(IID_NULL, &name, 1, LOCALE_USER_DEFAULT, &dwDispID);
    if (SUCCEEDED(hr))
    {
        // Get the property
        DISPPARAMS dispparamsNoArgs = {NULL, NULL, 0, 0};
        pDisp->Invoke(dwDispID, IID_NULL,LOCALE_USER_DEFAULT, DISPATCH_PROPERTYGET,
            &dispparamsNoArgs, &value, NULL, NULL);
        // The VARIANT value contains the value of the property

        // Sset the property
        VARIANT value2;
        VariantInit( value2);
        value2.vt= VT_UI1;
        value2.bval= 10;
```

```
        DISPPARAMS disparams;
        dispparams.rgvarg = &value2;
        DISPID dispidPut = DISPID_PROPERTYPUT;
        dispparams.rgdispidNamedArgs = &dispidPut;

        pDisp->Invoke(dwDispID, IID_NULL,LOCALE_USER_DEFAULT, DISPATCH_PROPERTYPUT,
            &dispparams, NULL, NULL, NULL);
    }
```

When the property is an `IUnknown*`,`IDispatch*`, or `SAFEARRAY*`, the flag `DISPATCH_PROPERTYPUTREF` must be used. This is also the case when a value is passed by reference (`VARIANT.vt = VT_BYREF | ...`).

The following example shows using the ATL helper it looks simple:

```
CComVariant prop;
CComDispatchDriver spDisp( pDisp);
// get the property
spDisp.GetPropertyByName(L"AttrByte",&prop);
//set the property
CComVariant newVal( (BYTE) 10);
spDisp.PutPropertyByName(L"AttrByte",&newVal);
```

The following example using VB and JScript it is simpler:

```
//VB
Dim prop As Byte
prop= obj.AttrByte

Dim newProp As Byte
newProp= 10
obj.AttrByte= newProp
'or
obj.AttrByte= 10

//JScript
var prop= obj.AttrByte;
obj.AttrByte= 10;
```

Service properties are not mapped to COM object properties. Use interfaces, such as `com.sun.star.beans.XPropertySet` to work with service properties.

Return Values

There are three possible ways to return values in UNO:

- function return values
- `inout` parameters
- `out` parameters

Return values are commonplace in most languages, whereas `inout` and `out` parameters are not necessarily supported. For example, in JScript.

To receive a return value in C++ provide a `VARIANT` argument to `IDispatch::Invoke`:

```
 //UNO IDL
long func();

//
 DISPPARAMS dispparams= { NULL, 0, 0, 0};
 VARIANT result;
 VariantInit( &result);
 hr= pdisp->Invoke( dispid, IID_NULL, LOCALE_USER_DEFAULT, DISPATCH_METHOD,
                          &dispparams, &result, NULL, 0);
```

The following example shows using VB and JScript this is simple:

```
//VB
Dim result As Long
result= obj.func

//JScript
var result= obj.func
```

When a function has `inout` parameters then provide arguments by reference in C++:

```
//UNO IDL
void func( [inout] long val);

//C++
long longOut= 10;
VARIANT var;
VariantInit(&var);
var.vt= VT_BYREF | VT_I4;
var.plVal= &longOut;

DISPPARAMS dispparams= { &var, 0, 1, 0};
hr= pdisp->Invoke( dispid, IID_NULL, LOCALE_USER_DEFAULT, DISPATCH_METHOD,
                &dispparams, NULL, NULL, 0);

//The value of longOut will be modified by UNO function.
```

The above VB code is written like this, because VB uses call by reference by default. After the call to `func()`, `value` contains the function output:

```
Dim value As Long
value= 10
obj.func value
```

The type of argument corresponds to the UNO type according to the default mapping, cf. *3.4.4 Professional UNO - UNO Language Bindings - Automation Bridge - Type Mappings*. If in doubt, use `VARIANT`s.

```
Dim value As Variant
value= 10;
obj.func value
```

However, there is one exception. If a function takes a character (`char`) as an argument and is called from VB, use an `Integer`, because there is no character type in VB. For convenience, the COM bridge also accepts a String as `inout` and `out` parameter:

```
//VB
Dim value As String
// string must contain only one character
value= "A"
Dim ret As String
obj.func value
```

JScript does not have `inout` or `out` parameters. As a workaround, the bridge accepts JScript `Array` objects. Index 0 contains the value.

```
// Jscript
var inout= new Array();
inout[0]=123;
obj.func( inout);
var value= inout[0];
```

Out parameters are similar to `inout` parameters in that the argument does not need to be initialized.

```
//C++
long longOut;
VARIANT var;
VariantInit(&var);
var.vt= VT_BYREF | VT_I4;
var.plVal= &longOut;

DISPPARAMS dispparams= { &var, 0, 1, 0};
hr= pdisp->Invoke( dispid, IID_NULL, LOCALE_USER_DEFAULT, DISPATCH_METHOD,
                &dispparams, NULL, NULL, 0);

//VB
Dim value As Long
obj.func value

//JScript
var out= new Array();
obj.func(out);
var value= out[0];
```

Usage of Types

Interfaces

Many UNO interface functions take interfaces as arguments. If this is the case, there are three possibilities to get an instance that supports the needed interface:

- Ask the service manager to create a service that implements that interface.
- Call a function on a UNO object that returns that particular interface.
- Provide an interface implementation if a listener object is required. Refer to *3.4.4 Professional UNO - UNO Language Bindings - Automation Bridge - Automation Objects with UNO Interfaces* for additional information.

If `createInstance()` is called on the service manager or another UNO function that returns an interface, the returned object is wrapped, so that it appears to be a COM dispatch object. When it is passed into a call to a UNO function then the original UNO object is extracted from the wrapper and the bridge makes sure that the proper interface is passed to the function. If UNO objects are used, UNO interfaces do not have to be dealt with. Ensure that the object obtained from a call to a UNO object implements the proper interface before it is passed back into another UNO call.

Structs

Automation does not know about structs as they exist in other languages, for example, in C++. Instead, it uses Automation objects that contain a set of properties similar to the fields of a C++ struct. Setting or reading a member ultimately requires a call to `IDispatch::Invoke`. However in languages, such as VB, VBScript, and JScript, the interface call is obscured by the programming language. Accessing the properties is as easy as with C++ structs.

```
// VB. obj is an object that implements a UNO struct
obj.Width= 100
obj.Height= 100
```

Whenever a UNO function requires a struct as an argument, the struct must be obtained from the UNO environment. It is not possible to declare a struct. For example, assume there is an office function `setSize()` that takes a struct of type `Size`. The struct is declared as follows:

```
// UNO IDL
struct Size
{
    long Width;
    long Height;
}

// the interface function, that will be called from script
void XShape::setSize( Size aSize)
```

You cannot write code similar to the following example (VBScript):

```
Class Size
    Dim Width
    Dim Height
End Class

'obtain object that implements Xshape

'now set the size
call objXShape.setSize( new Size) // wrong
```

The `com.sun.star.reflection.CoreReflection` service or the `Bridge_GetStruct` function that is called on any UNO object can be used to create the struct. The following example uses the `CoreReflection` service

```
'VBScript in Windows Scripting Host
Set objServiceManager= Wscript.CreateObject("com.sun.star.ServiceManager")

'Create the CoreReflection service that is later used to create structs
Set objCoreReflection= objServiceManager.createInstance("com.sun.star.reflection.CoreReflection")
```

```
'get a type description class for Size
Set classSize= objCoreReflection.forName("com.sun.star.awt.Size")
'create the actual object
Dim aSize
classSize.createObject aSize
'use aSize
aSize.Width= 100
aSize.Height= 12

'pass the struct into the function
objXShape.setSize aSize
```

The next example shows how `Bridge_GetStruct` is used.

```
Set objServiceManager= Wscript.CreateObject("com.sun.star.ServiceManager")
Set aSize= objServiceManager.Bridge_GetStruct("com.sun.star.awt.Size")
'use aSize
aSize.Width= 100
aSize.Height= 12

objXShape.setSize aSize
```

The `Bridge_GetStruct` function can be called on any UNO object, as well as the service manager.

The corresponding C++ examples look complicated, but ultimately the same steps are necessary. The method `forName()` on the `CoreReflection` service is called and returns a `com.sun.star.reflection.XIdlClass` which can be asked to create an instance using `createObject()`:

```
// create the service manager of OpenOffice
IDispatch* pdispFactory= NULL;
CLSID clsFactory= {0x82154420,0x0FBF,0x11d4,{0x83, 0x13,0x00,0x50,0x04,0x52,0x6A,0xB4}};
hr= CoCreateInstance( clsFactory, NULL, CLSCTX_ALL, __uuidof(IDispatch), (void**)&pdispFactory);

// create the CoreReflection service
OLECHAR* funcName= L"createInstance";
DISPID id;
pdispFactory->GetIDsOfNames( IID_NULL, &funcName, 1, LOCALE_USER_DEFAULT, &id);

VARIANT param1;
VariantInit( &param1);
param1.vt= VT_BSTR;
param1.bstrVal=   SysAllocString( L"com.sun.star.reflection.CoreReflection");
DISPPARAMS dispparams= { &param1, 0, 1, 0};
VARIANT result;
VariantInit( &result);
hr= pdispFactory->Invoke( id, IID_NULL, LOCALE_USER_DEFAULT, DISPATCH_METHOD,
                          &dispparams, &result, NULL, 0);
IDispatch* pdispCoreReflection= result.pdispVal;
pdispCoreReflection->AddRef();
VariantClear( &result);

// create the struct's idl class object
OLECHAR* strforName= L"forName";
hr= pdispCoreReflection->GetIDsOfNames( IID_NULL, &strforName, 1, LOCALE_USER_DEFAULT, &id);
VariantClear( &param1);
param1.vt= VT_BSTR;
param1.bstrVal= SysAllocString(L"com.sun.star.beans.PropertyValue");
hr= pdispCoreReflection->Invoke( id, IID_NULL, LOCALE_USER_DEFAULT,
                                 DISPATCH_METHOD, &dispparams, &result, NULL, 0);

IDispatch* pdispClass= result.pdispVal;
pdispClass->AddRef();
VariantClear( &result);

// create the struct
OLECHAR* strcreateObject= L"createObject";
hr= pdispClass->GetIDsOfNames( IID_NULL,&strcreateObject, 1, LOCALE_USER_DEFAULT, &id)

IDispatch* pdispPropertyValue= NULL;
VariantClear( &param1);
param1.vt= VT_DISPATCH | VT_BYREF;
param1.ppdispVal= &pdispPropertyValue;
hr= pdispClass->Invoke( id, IID_NULL, LOCALE_USER_DEFAULT,
                DISPATCH_METHOD, &dispparams, NULL, NULL, 0);

// do something with the struct pdispPropertyValue contained in dispparams
// ...

pdispPropertyValue->Release();
pdispClass->Release();
pdispCoreReflection->Release();
pdispFactory->Release();
```

The `Bridge_GetStruct` example.

```
// object be some UNO object in a  COM environment
OLECHAR* strstructFunc= L"Bridge_GetStruct";
hr= object->GetIDsOfNames( IID_NULL, &strstructFunc, 1, LOCALE_USER_DEFAULT, &id);

VariantClear(&result);
VariantClear( &param1);
param1.vt= VT_BSTR;
param1.bstrVal= SysAllocString(
L"com.sun.star.beans.PropertyValue");
hr= object->Invoke( id, IID_NULL,LOCALE_USER_DEFAULT, DISPATCH_METHOD,
                        &dispparams, &result, NULL, 0);

IDispatch* pdispPropertyValue= result.pdispVal;
pdispPropertyValue->AddRef();

// do something with the struct pdispPropertyValue
...
```

JScript:

```
// struct creation via CoreReflection
var objServiceManager= new ActiveXObject("com.sun.star.ServiceManager");
var objCoreReflection= objServiceManager.createInstance("com.sun.star.reflection.CoreReflection");

var classSize= objCoreReflection.forName("com.sun.star.awt.Size");
var outParam= new Array();
classSize.createObject( outParam);
var size= outParam[0];
//use the struct
size.Width=111;
size.Height=112;
// -----------------------------------------------------
// struct creation by bridge function
var objServiceManager= new ActiveXObject("com.sun.star.ServiceManager");
var size= objServiceManager.Bridge_GetStruct("com.sun.star.awt.Size");
size.Width=111;
size.Height=112;
```

Type Mappings

Mapping of Simple Types

Whenever a UNO interface function requires a value of a simple type, such as `float`, `double`, `byte`, `short`, `long` or `char`, it is provided by declaring a variable of that type (or a constant or temporary variable) in the programming language used and passes it as an argument. This is the customary way of programming. UNO simple types are the same as Automation types and the bridge has a method of converting them.

In some languages, the set of available types does not match those of UNO types. For instance, in Visual Basic, a character can not be declared, and a string is used instead. This does not concern a VB programmer, but if C++ is used, then you would typically provide a short value ('A') which is totally different from the `BSTR` string that is used when you write "A" in VB.

Other examples for languages with a different set of simple types (compared to UNO) are the scripting languages VBScript and JScript. They are considered to be type-less languages, because they do not allow variables of specific types to be declared. At a basic level they use Automation types as well. They may not use the whole range (this is an implementation detail and might differ between scripting engines). For example, they use a `double` for every floating point value and a `signed long` for all integer values. This does not pose a problem, because the bridge converts those values into the expected floating point or integer types. The programmer has to be aware of this fact to prevent unexpected results caused by providing a value that exceeds the range of the expected UNO type. For example, if you pass an integer value of 65536 where the UNO type is a `byte` (-128 to 127), the converted value is different then the one provided. Also the conversion of `double` to `float` or vice versa often results in slightly different values.

Automatic Type Conversion

Every UNO object obtained directly or indirectly from the service manager is an Automation object that contains the actual UNO object. The wrapper contains code that implements the `IDispatch` interface. During a call to `IDispatch::Invoke`, the wrapper-code converts the arguments to UNO values, and the respective UNO function of the contained object is called with those values as arguments.

The `IDispatch` interface reveals that all arguments and return values are actually `VARIANT`s. This is similar to the `XInvocation` interface where arguments and return values are of the `any` type. Both types carry values of a specific type. A `VARIANT` can contain all Automation types and an any can contain all UNO types. Therefore, it would be suitable to say that `VARIANT` values are converted into `any` values and vice versa. The contained values still have to be converted. When working with `VARIANT`s and `any`s, extract the contained values for further processing. Before calling those interfaces, put the values into `VARIANT`s or `any`s. This process is sometimes hidden by the programming language you use. For example:

```
//UNO IDL
string func([in] long value);
//VB
Dim value As Long
value= 100
Dim ret As String
ret= obj.func( value)
```

Since `VARIANT` and `any` are helper types that allow writing code when the specific types are not yet known, we need to focus on the mapping of the specific types. The `VARIANT`s or `any`s are only mentioned if there is reason to look beyond the mapping of the contained types.

The bridge converts arguments according to well-defined mappings. The default mappings are sufficient in most cases. The bridge also accepts arguments for flexibility whose types do not exactly match the default mappings, but are similar enough to be converted (*3.4.4 Professional UNO - UNO Language Bindings - Automation Bridge - Type Mappings*). In some situations, it may be necessary for an Automation client to specify how an argument should be treated. This can be the case in scripting languages, where the language does not provide specific types. Refer to *3.4.4 Professional UNO - UNO Language Bindings - Automation Bridge - Type Mappings*. When Automation objects are used from UNO, there is no construct as a `Value Object`. The bridge always uses the default mappings.

Default Mappings from Automation Types to UNO

This mapping applies in two situations. First, whenever you call a UNO function from an Automation environment, for example, from VB, the arguments flagged as in or inout parameters in the corresponding UNO IDL description are converted according to the following default mappings. Second, when Automation objects are called from a UNO environment and return a value, the return values are converted to the corresponding UNO types.

Automation IDL Types (source)	**UNO IDL Types (target)**
`boolean`	`boolean`
`unsigned char`	`byte`
`double`	`double`
`float`	`float`
`short`	`short`
`long`	`long`
`BSTR`	`string`
`short`	`char`

Automation IDL Types (source)	UNO IDL Types (target)
`long`	`enum`
`IDispatch*`	The IDispatch* is mapped to the expected interface if it is actually a UNO object implementing that interface, or an Automation implementation of that interface (see *3.4.4 Professional UNO - UNO Language Bindings - Automation Bridge - Usage of Types*). If the IDispatch* is a return value or out parameter, then it is mapped to `XInvocation`. If the Automation object is a struct, UNO receives a struct (see *3.4.4 Professional UNO - UNO Language Bindings - Automation Bridge - Usage of Types*).
`SAFEARRAY` or `IDispatch*` in JScript	`sequence< type >`. A two-dimensional `SAFEARRAY` is converted to `sequence< sequence<type>>`, a three-dimensional `SAFEARRAY` is converted to `sequence<sequence<sequence<type>>>`, and so forth. In JScript one would provide an `Array` object that contains other `Array` objects.

Default Mappings from UNO Types to Automation

If an Automation client calls a function on a UNO object and the function returns values such as out parameters or the return value, then the mapping from UNO to Automation types applies. For example:

```
//UNO IDL
long func([out] long value);

//call from VB
Dim value As Long
Dim ret As Long
ret= obj.func(value)
```

The returned value is a UNO `long` that is converted into an Automation `long`, which fits the declaration of the variable `ret`. When the UNO function returns, then the bridge converts the out parameter according to the mapping and writes the value back into the variable `value`.



In some situations, the client code performs a conversion on its own (this behavior is covered in "Client-Side Conversions").

This mapping is also used when you pass arguments to functions on an Automation object from a UNO environment.

UNO IDL Types (source)	Automation IDL Types (target)
`boolean`	`boolean`
`char`	`short`
`byte`	`unsigned byte`
`double`	`double`
`float`	`float`
`short, unsigned short`	`short`
`long, unsigned long`	`long`
`string`	`BSTR`
`interface,struct`	`IDispatch*`
`sequence`	`SAFEARRAY(VARIANT)`

If a UNO function returns interfaces or structs, they are converted into Automation objects. For example:

```
//UNO IDL
void func([out]com.sun.star.lang.XEventListener aInterface, [out]com.sun.star.lang.EventObject aStruct);

//VB
Dim objEventListener As Object
Dim objStruct As Object
func objEventListener, objStruct
```

A `sequence` returned by a UNO function is converted into a `SAFEARRAY` that contains `VARIANT`s. If a sequence contains nested sequences, the `VARIANT`s contain `SAFEARRAY`s. The `OleObjectFactory` creates Automation objects and provides an `com.sun.star.script.XInvocation` interface which can be used from the UNO environment. These objects might expect multi-dimensional `SAFEARRAY`s as arguments. In this is the case, provide an appropriate sequence, for example `sequence<sequence<long>>` for a two-dimensional array of longs. The contained sequences should have the same length, otherwise the bridge uses the longest sequence to stipulate the size of the respective dimension. If a sequence is shorter, then the remaining values are filled with null values.

For example, assume a sequence with two elements that are sequences of ten elements. The elements of the two sequences are long types and the first sequence could be mapped to an array, which is expressed in C (for convenience):

```
long ar[2][10];
```

Further assume that the second of the two contained sequences only contains five elements. The C array would still look the same. With the difference, that `ar[0][0]` through `ar[0][4]` contain the elements of that sequence, and `ar[0][4]` through `ar[0][9]` contain null values.

Sequences are mapped to `SAFEARRAY`s and not C arrays.

Conversion Mappings

As shown in the previous section, Automation types have a UNO counterpart according to the mapping tables. If a UNO function expects a particular type as an argument, then supply the corresponding Automation type. This is not always necessary as the bridge also accepts similar types. For example:

```
//UNO IDL
void func( long value);
// VB
Dim value As Byte
value = 2
obj.func valLong
```

The following table shows the various Automation types, and how they are converted to UNO IDL types if the expected UNO IDL type has not been passed.

Automation IDL Types (source)	**UNO IDL Types (target)**
boolean (true, false) unsigned char, short, long, float, double: 0 = false, > 0 = true string: "true" = true, "false" = false	boolean
boolean, unsigned char, short, long, float, double, string	byte
double, boolean, unsigned char, short, long, float, string	double
float, boolean, unsigned char, short, string	float
short, unsigned char, long, float, double, string	short
long, unsigned char, long, float, double, string	long
BSTR, boolean, unsigned char, short, long, float, double	string
short, boolean, unsigned char, long, float, double, string (1 character long)	char
long, boolean, unsigned char, short, float, double, string	enum

When you use a string for a numeric value, it must contain an appropriate string representation of that value.

Floating point values are rounded if they are used for integer values.

Be careful using types that have a greater value space than the UNO type. Do not provide an argument that exceeds the value space which would result in an error. For example:

```
// UNO IDL
void func([in] byte value);
```

```
// VB
Dim value as Integer
value= 1000
obj.func value  'causes an error
```

The conversion mappings only work with in parameters, that is, during calls from an Automation environment to a UNO function, as far as the UNO function takes in parameters.

Client-Side Conversions

The UNO IDL description and the defined mappings indicate what to expect as a return value when a particular UNO function is called. However, the language used might apply yet another conversion after a value came over the bridge.

```
// UNO IDL
float func();
```

```
// VB
Dim ret As Single
ret= obj.func() 'no conversion by VB

Dim ret2 As String
ret2= obj.func() 'VB converts float to string
```

When the function returns, VB converts the `float` value into a `string` and assigns it to `ret2`. Such a conversion comes in useful when functions return a character, and a string is preferred instead of a VB Integer value.

```
// UNO IDL
char func();
```

```
// VB
Dim ret As String
ret= obj.func()  'VB converts the returned short into a string
```

Be aware of the different value spaces if taking advantage of these conversions. That is, if the value space of a variable that receives a return value is smaller than the UNO type, a runtime error might occur if the value does not fit into the provided variable. Refer to the documentation of your language for client-side conversions.

Client-side conversions only work with return values and not with out or inout parameters. The current bridge implementation is unable to transport an out or inout parameter back to Automation if it does not have the expected type according to the default mapping.

Another kind of conversion is done implicitly. The user has no influence on the kind of conversion. For example, the scripting engine used with the Windows Scripting Host or Internet Explorer uses `double` values for all floating point values. Therefore, when a UNO function returns a `float` value, then it is converted into a `double` which may cause a slightly different value. For example:

```
// UNO IDL
float func();  //returns 3.14
```

```
// JScript
var ret= obj.func(); // implicit conversion from float to double, ret= 3.14000010490417
```

Mapping of Any

The any type is similar to VARIANT in COM. That is, it can contain values of different types. Do not put a value into a VARIANT if a function argument needs to be an any.

```
// UNO IDL
interface XSomething: XInterface
{
    void func([in] any value);
};
```

```
// Visual Basic
Dim param As Long
param= 10

// obj is the object that implements XSomething
obj.func param
```

In C++, set the value directly in the VARIANT that is put into the DISPPARAMS.rgvarg array. That is, there is no need to provide a VARIANT with the type VT_VARIANT | VT_BYREF.

Although an any can contain all possible UNO types, an any argument must contain a certain type. An example is the com.sun.star.beans.XPropertySet interface with its function:

```
// UNO IDL
void setPropertyValue( [in] string aPropertyName,
         [in] any aValue ) raises ...
```

As the name suggests, the function is used to set a value for a particular property. Usually the properties have a distinct type and are not anys. Lets assume that there is a property PropA of type float. Then a Single in VB or a float in C++ has to be provided. In JScript or VBScript, the scripting engine will probably pass a double to the function which would not be converted by the bridge. That is, setPropertyValue() would receive an any containing a double. If the programmer of the XPropertySet implementation was not careful converting the any into the type that is expected then the code will throw an exception. There is no rule about how tolerant the implementation has to be. The bridge does not know that the property is a float and hence it needs to be told. This it is done by providing a Value Object as argument. A Value Object is an Automation object that is provided by the bridge. It carries a value and the name of the type that it is supposed to be. For example:

```
// VBScript with Windows Scripting Host (WSH)
Set objServiceManager= WScript.CreateObject("com.sun.star.ServiceManager")
Set aFloat= objServiceManager.Bridge_GetValueObject()
aFloat.Set "float", 3.14

// obj is the implementation of XSomething
obj.setPropertyValue "PropA", aFloat
```

Value Objects are covered in depth in chapter *3.4.4 Professional UNO - UNO Language Bindings - Automation Bridge - Type Mappings*.

In JScript, an any argument can cause problems when the client provides an array or object as a parameter, because the Array object is used for arrays. The bridge receives an IDispatch pointer in both cases, because it is an object. To make a decision about what conversion is applied, the bridge obtains type information about the function that is to be called and proceeds accordingly:

```
// UNO IDL
void func( XSomething obj);
void func2( sequence<long> ar);
```

The bridge has to convert an IDispatch in both cases.,The type information says exactly what IDispatch* supports: It must be an interface for func() , but a sequence for func2().

```
// UNO IDL
void func( any val);
```

```
// JScript
// obj is an automation object
func( obj);
```

The bridge obtains type information in the above example, but it only knows that the argument is of type `any`. The bridge does not know whether to convert `IDispatch` into an interface or a sequence. Since the conversions are different, a wrong decision will cause an error when the converted object is accessed later. In this situation, the bridge assumes that the object is an array if it has a property named `0`. This will work, because an interface has a `get0()` or `set0()` function. If not, use a `Value Object` for arrays and objects which could contain a `0` property.

Mapping of String

A string is a data structure that is common in programming languages. Although the idea of a string is the same, the implementations and their creation can be quite different. For example, a C++ programmer has a range of possibilities to choose from (for example, `char*`, `char[ ]`, `wchar_t*`, `wchar_t[ ]`, `std::string`, `CString`, `BSTR`), where a JScript programmer only knows one kind of string. To use Automation across languages, it is necessary to use a string type that is common to all those languages that has the same binary representation. This particular string is declared as `BSTR` in COM. The name can be different depending on the language. For example, in C++ there is a `BSTR` type, in VB it is called `String` and in JScript every string defined is a `BSTR`. Refer to the documentation covering the `BSTR`'s equivalent if using an Automation capable language not covered by this document.

Mapping of Sequence

The discussion about strings applies for arrays as well. The difference is the array type used by Automation is named `SAFEARRAY` in COM. The `SAFEARRAY` array is to be used when a UNO function takes a sequence as an argument. To create a `SAFEARRAY` in C++, use Windows API functions. The C++ name is also `SAFEARRAY`, but in other languages it might be named different. In VB for example, the type does not even exist, because it is mapped to an ordinary VB array:

```
Dim myarr(9) as String
```

JScript is different. It does not have a method to create a `SAFEARRAY`. Instead, JScript features an `Array` object that can be used as a common array in terms of indexing and accessing its values. It is represented by a dispatch object internally. JScript offers a `VBArray` object that converts a `SAFEARRAY` into an `Array` object. Therefore, it is possible to call functions on Automation objects which return `SAFEARRAY`s.

When a `SAFEARRY` is provided and a function is expecting a UNO sequence, the bridge accepts JScript `Array` objects and converts them into a UNO sequence.



If a `SAFEARRAY` is obtained in JScript as a result of a call to an ActiveX component or a VB Script function (for example, the Internet Explorer allows JScript and VBS code on the same page), then it can also be used as an argument of a UNO function without converting it to an `Array` object.

If a UNO function returns a sequence, a `SAFEARRAY` is returned in JScript. Use the `VBArray` object to convert the `SAFEARRAY` into a JScript `Array to` process the array.

Value Objects

Since the Automation bridge supports JScript, it has to deal with ambiguities when it comes to the conversion of arguments. Arguments, which are dispatch objects, can represent three different kinds of values: objects, arrays or out/ inout parameters. To solve this problem, the bridge obtains type information about the UNO function that receives the argument. The bridge must always get type information if an argument is an object, because the bridge does not know the language that is being used. In a remote environment, this causes additional network roundtrips and slows down overall performance. With a `Value Object`, programmers can provide some type information and save the bridge from obtaining it.

When a UNO interface function takes an `any` as an argument and the bridge receives an object, then it be mistaken for a JScript array. This is described in paragraph *3.4.4 Professional UNO - UNO Language Bindings - Automation Bridge - Type Mappings*. With a `Value Object`, the bridge knows exactly what the object stands for and converts it correctly.

A `Value Object` is an Automation object. It offers functions for setting and getting a value, and determines if it represents an inout or out parameter. The client can use `Value Object`s for every possible argument of a UNO function.

A `Value Object` exposes four functions that can be accessed through `IDispatch`. These are:

```
void Set( [in]VARIANT type, [in]VARIANT value);
```

Assigns a type and a value.

```
void Get( [out,retval] VARIANT* val);
```

Returns the value contained in the object. Get is used when the `Value Object` was used as inout or out parameter.

```
void InitOutParam();
```

Tells the object that it is used as out parameter.

```
void InitInOutParam( [in]VARIANT type, [in]VARIANT value);
```

Tells the object that it is used as inout parameter and passes the value for the in parameter, as well as the type.

When the `Value Object` is used as in or inout parameter then specify the type of the value. The names of types correspond to the names used in UNO IDL, except for the “object” name. The following table shows what types can be specified.

Name (used with Value Object)	**UNO IDL**
char	char
boolean	boolean
byte	byte
unsigned	unsigned byte
short	short
unsigned short	unsigned short
long	long
unsigned long	unsigned long
string	string
float	float
double	double
any	any
object	some UNO interface

To show that the value is a sequence, put brackets before the names, for example:

```
[]char - sequence<char>
[][]char - sequence < sequence <char > >
[][][]char - sequence < sequence < sequence < char > > >
```

The `Value Object`s are provided by the bridge and can be obtained from UNO objects. Call the function `Bridge_GetValueObject`:

```
// object is some UNO wrapper object
var valueObject= object.Bridge_GetValueObject();
```

To use a `Value Object` as in parameter, specify the type and pass the value to the object:

```
// UNO IDL
void doSomething( [in] sequence< short > ar);
```

```
// JScript
var value= object.Bridge_GetValueObject();
var array= new Array(1,2,3);
value.Set("[]short",array);
object.doSomething( value);
```

In the previous example, the `Value Object` was defined to be a sequence of short values. The array could also contain `Value Object`s again:

```
var value1= object.Bridge_GetValueObject();
var value2= object.Bridge_GetValueObject();
value1.Set("short", 100);
value2.Set("short", 111);
var array= new Array();
array[0]= value1;
array[1]= value2;
var allValue= object.Bridge_GetValueObject();
allValue.Set("[]short", array);
object.doSomething( allValue);
```

If a function takes an out parameter, tell the `Value Object` like this:

```
// UNO IDL
void doSomething( [out] long);
```

```
// JScript
var value= object.Bridge_GetValueObject();
value.InitOutParam();
object.doSomething( value);
var out= value.Get();
```

When the `Value Object` is an inout parameter, it needs to know the type and value as well:

```
//UNO IDL
void doSomething( [inout] long);
```

```
//JScript
var value= object.Bridge_GetValueObject();
value.InitInOutParam("long", 123);
object.doSomething(value);
var out= value.Get();
```

Exceptions and Errorcodes

UNO interface functions may throw exceptions to communicate an error. Automation objects provide a different error mechanism. First, the `IDispatch` interface describes a number of error codes (`HRESULT`s) that are returned under certain conditions. Second, the `Invoke` function takes an argument that can be used by the object to provide descriptive error information. The argument is a structure of type `EXCEPINFO` and is used by the bridge to convey exceptions being thrown by the called UNO interface function. In case of an exception, the bridge fills in the following values:

`EXCEPINFO::wCode` = 1001

`EXCEPINFO::bstrSource` = “any ONE component”

`EXCEPINFO::bstrDescription` = type name of the exceptions

If the caller does not provide an `EXCEPINFO` argument, then `Invoke` returns a `DISP_E_EXCEPTION` as `HRESULT`.

As already stated, the functions of `IDispatch` return error codes. The reasons for those codes are shown in the following tables.

Possible `HRESULT` return values of `IDispatch::Invoke` are:

HRESULT	Reason
DISP_E_EXCEPTION	• UNO interface function or property access function threw an exception and the caller did not provide an EXCEPINFO argument. • Bridge error. A ValueObject could not be created when the client called Bridge_GetValueObject. • Bridge error. A struct could not be created when the client called Bridge_GetStruct • Bridge error. The automation object contains a UNO object that does not support the XInvocation interface. Could be a failure of com.sun.star.script.Invocation service. • In JScript was an Array object passed as inout param and the bridge could not retrieve the property "0". • A conversion of a VARIANTARG (DISPPARAMS structure) failed for some reason. • Parameter count does not tally with the count provided by UNO type information (only when one DISPPARAMS contains VT_DISPATCH). This is a bug. DISP_E_BADPARAMCOUNT should be returned.
DISP_E_NONAMEDARGS	• The caller provided "named arguments" for a call to a UNO function.
DISP_E_BADVARTYPE	• Conversion of VARIANTARGs failed. • Bridge error: Caller provided a ValueObject and the attempt to retrieve the value failed. This is possibly a bug. DISP_E_EXCEPTION should be returned. • A member with the current name does not exist according to type information. This is a bug. DISP_E_MEMBERNOT-FOUND should be returned.
DISP_E_BADPARAMCOUNT	• A property was assigned a value and the caller provided null or more than one arguments. • The caller did not provide the number of arguments as required by the UNO interface function.
DISP_E_MEMBERNOTFOUND	• Invoke was called with a DISPID that was not issued by GetIDsOfName (OleBridgeSupplier2) • There is no interface function (also property access function) with the name for which Invoke is currently being called.
DISP_E_TYPEMISMATCH	The called provided an argument of a false type.
DISP_E_OVERFLOW	An argument could not be coerced to the expected type. Internal call to XInvocation::invoke resulted in a CannotConvertException being thrown. The field reason has the value OUT_OF_RANGE which means that a given value did not fit in the range of the destination type.

HRESULT	Reason
E_UNEXPECTED	[2]results from com.sun.star.script.CannotConvertException of XInvocation::invoke with FailReason::UNKNOWN. Internal call to XInvocation::invoke resulted in a com.sun.star.script.CannotConvertException being thrown. The field reason has the value UNKNOWN, which signifies some unknown error condition.
E_POINTER	Bridge_GetValueObject or Bridge_GetStruct called and no argument for return value provided.
S_OK	Ok.

Return values of IDispatch::GetIDsOfNames:

HRESULT	Reason
E_POINTER	Caller provided no argument that receives the DISPID.
DISP_E_UNKNOWNNAME	There is no function or property with the given name. OleBridgeSupplierVar1: The name has been determined not to exist by a previous call to IDispatch::Invoke
S_OK	Ok.

The functions IDispatch::GetTypeInfo and GetTypeInfoCount return E_NOTIMPL.

When a call from UNO to an Automation object (OleObjectFactory) is performed, then the following HRESULT values are converted to exceptions. Keep in mind that it is determined what exceptions the functions of XInvocation are allowed to throw.

Exceptions thrown by XInvocation::invoke() and their HRESULT counterparts:

HRESULT	Exception
DISP_E_BADPARAMCOUNT	com.sun.star.lang.IllegalArgumentException
DISP_E_BADVARTYPE	com.sun.star.uno.RuntimeException
DISP_E_EXCEPTION	com.sun.star.reflection.InvocationTargetException
DISP_E_MEMBERNOTFOUND	com.sun.star.lang.IllegalArgumentException
DISP_E_NONAMEDARGS	com.sun.star.lang.IllegalArgumentException
DISP_E_OVERFLOW	com.sun.star.script.CannotConvertException, reason= FailReason::OUT_OF_RANGE
DISP_E_PARAMNOTFOUND	com.sun.star.lang.IllegalArgumentException
DISP_E_TYPEMISMATCH	com.sun.star.script.CannotConvertException, reason= FailReason::UNKNOWN
DISP_E_UNKNOWNINTERFACE	com.sun.star.uno.RuntimeException
DISP_E_UNKNOWNLCID	com.sun.star.uno.RuntimeException
DISP_E_PARAMNOTOPTIONAL	com.sun.star.script.CannotConvertException, reason= FailReason::NO_DEFAULT_AVAILABLE

XInvocation::setValue() throws the same as invoke() except for:

HRESULT	Exception
DISP_E_BADPARAMCOUNT	com.sun.star.uno.RuntimeException
DISP_E_MEMBERNOTFOUND	com.sun.star.beans.UnknownPropertyException
DISP_E_NONAMEDARGS	com.sun.star.uno.RuntimeException

`XInvocation::getValue()` throws the same as `invoke()` except for:

HRESULT	Exception
DISP_E_BADPARAMCOUNT	com.sun.star.uno.RuntimeException
DISP_E_EXCEPTION	com.sun.star.uno.RuntimeException
DISP_E_MEMBERNOTFOUND	com.sun.star.beans.UnknownPropertyException
DISP_E_NONAMEDARGS	com.sun.star.uno.RuntimeException
DISP_E_OVERFLOW	com.sun.star.uno.RuntimeException
DISP_E_PARAMNOTFOUND	com.sun.star.uno.RuntimeException
DISP_E_TYPEMISMATCH	com.sun.star.uno.RuntimeException
DISP_E_PARAMNOTOPTIONAL	com.sun.star.uno.RuntimeException

Automation Objects with UNO Interfaces

It is common that UNO functions take interfaces as arguments. As discussed in section *3.4.4 Professional UNO - UNO Language Bindings - Automation Bridge - Usage of Types*, those objects are usually obtained as return values of UNO functions. With the Automation bridge, it is possible to implement those objects even as Automation objects and use them as arguments, just like UNO objects.

Although Automation objects can act as UNO objects, they are still not fully functional UNO components. That is, they cannot be created by means of the service manager.



However, that feature may be implemented in the future. The factories and how they map to COM class factories will have to be considered. Also a loader is needed (components from different environments have different loaders). The loader, however, could make use of the `OleObjectFactory` service.

One use case for such objects are listeners. For example, if a client wants to know when a writer document is being closed, it can register the listener object with the document, so that it will be notified when the document is closing.

Requirements

Automation objects implement the `IDispatch` interface, and all function calls and property operations go through this interface. We imply that all interface functions are accessed through the dispatch interface when there is mention of an Automation object implementing UNO interfaces. That is, the Automation object still implements `IDispatch` only.

Basically, all UNO interfaces can be implemented as long as the data types used with the functions can be mapped to Automation types. The bridge needs to know what UNO interfaces are supported by an Automation object, so that it can create a UNO object that implements all those interfaces. This is done by requiring the Automation objects to support the property `Bridge_implementedInterfaces`, which is an array of strings. Each of the strings is a fully qualified name of an implemented interface. If an Automation object only implements one UNO interface, then it does not need to support that property.



You never implement `com.sun.star.script.XInvocation` and `com.sun.star.uno.XInterface`. `XInvocation` cannot be implemented, because the bridge already maps `IDispatch` to `XInvocation` internally. Imagine a function that takes an `XInvocation`:

```
// UNO IDL
void func( [in] com.sun.star.script.XInvocation obj);
```

In this case, use any Automation object as argument. When an interface has this function,

```
void func( [in] com.sun.star.XSomething obj)
```

the automation object must implement the functions of `XSomething`, so that they can be called through IDispatch::Invoke.

Examples

The following example shows how a UNO interface is implemented in VB. It is about a listener that gets notified when a writer document is being closed.

To rebuild the project use the wizard for an ActiveX *dll* and put this code in the class module. The component implements the `com.sun.star.lang.XEventListener` interface.

```
Option Explicit
Private interfaces(0) As String

Public Property Get Bridge_ImplementedInterfaces() As Variant
    Bridge_ImplementedInterfaces = interfaces
End Property

Private Sub Class_Initialize()
interfaces(0) = "com.sun.star.lang.XEventListener"
End Sub

Private Sub Class_Terminate()
    On Error Resume Next
    Debug.Print "Terminate VBEventListener"
End Sub

Public Sub disposing(ByVal source As Object)
    MsgBox "disposing called"
End Sub
```

You can use these components in VB like this:

```
Dim objServiceManager As Object
Dim objDesktop As Object
Dim objDocument As Object
Dim objEventListener As Object

Set objServiceManager= CreateObject("com.sun.star.ServiceManager")
Set objDesktop= objServiceManager.createInstance("com.sun.star.frame.Desktop")

'Open a new empty writer document
Dim args()
Set objDocument= objDesktop.loadComponentFromURL("private:factory/swriter", "_blank", 0, args)
'create the event listener ActiveX component
Set objEventListener= CreateObject("VBasicEventListener.VBEventListener")

'register the listener with the document
objDocument.addEventListener objEventlistener
```

The next example shows a JScript implementation of a UNO interface and its usage from JScript. To use JScript with UNO, a method had to be determined to realize arrays and out parameters. Presently, if a UNO object makes a call to a JScript object, the bridge must be aware that it has to convert arguments according to the JScript requirements. Therefore, the bridge must know that one calls a JScript component, but the bridge is not capable of finding out what language was used. The programmer has to provide hints, by implementing a property with the name "`_environment`"that has the value "`JScript`".

```
// UNO IDL: the interface to be implemented
interface XSimple : public com.sun.star.uno.XInterface
{
    void func1( [in] long val, [out] long outVal);
    long func2( [in] sequence< long > val, [out] sequence< long > outVal);
    void func3( [inout]long);
```

```
};
```

```
// JScript: implementation of XSimple
function XSimplImpl()
{
    this._environment= "JScript";
    this.Bridge_implementedInterfaces= new Array( "XSimple");

    // the interface functions
    this.func1= func1_impl;
    this.func2= func2_impl;
    this.func3= func3_impl;
}

function func1_impl( inval, outval)
{
    //outval is an array
    outval[0]= 10;
    ...
}

function func2_impl(inArray, outArray)
{
    outArray[0]= inArray;
    // or
    outArray[0]= new Array(1,2,3);

    return 10;
}

function func3_impl(inoutval)
{
    var val= inoutval[0];
    inoutval[0]= val+1;
}
```

Assume there is a UNO object that implements the following interface function:

```
//UNO IDL
void doSomething( [in] XSimple);
```

Now, call this function in JScript and provide a JScript implementation of `XSimple`:

```
<script language="JScript">

var factory= new ActiveXObject("com.sun.star.ServiceManager");
// create the UNO component that implements an interface with the doSomething function
var oletest= factory.createInstance("oletest.OleTest");
oletest.doSomething( new XSimpleImpl());
...
```

To build a component with C++, write the component from scratch or use a kind of framework, such as the Active Template Library (ATL). When a dual interface is used with ATL, the implementation of `IDispatch` is completely hidden and the functions must be implemented as if they were an ordinary custom interface, that is, use specific types as arguments instead of `VARIANT`s. If a UNO function has a return value, then it has to be specified as the first argument which is flagged as “retval”.

```
</script>
// UNO IDL
interface XSimple : public com.sun.star.uno.XInterface
{
    void func1( [in] long val, [out] long outVal);
    long func2( [in] sequence< long > val, [out] sequence< long > outVal);
};

//IDL of ATL component
[
    object,
    uuid(xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx),
    dual,
    helpstring("ISimple Interface"),
    pointer_default(unique)
]
interface ISimple : IDispatch
{
    [id(1), helpstring("method func1")]
                HRESULT func1( [in] long val, [out] long* outVal);
    [id(2), helpstring("method func2")]
                HRESULT func2([out,retval] long ret, [in] SAFEARRAY(VARIANT) val,
                        [out] SAFEARRAY(VARIANT) * outVal);
```

```
    [propget, id(4), helpstring("property_implementedInterfaces")]
        HRESULT Bridge_implementedInterfaces([out, retval] SAFEARRAY(BSTR) *pVal);
};
```

DCOM

The Automation bridge maps all UNO objects to automation objects. That is, all those objects implement the `IDispatch` interface. To access a remote interface, the client and server must be able to marshal that interface. The marshaling for `IDispatch` is already provided by Windows, therefore all objects which originate from the bridge can be used remotely.

To make DCOM work, apply proper security settings for client and server. This can be done by setting the appropriate registry entries or programmatically by calling functions of the security API within the programs. The office does not deal with the security, hence the security settings can only be determined by the registry settings which are not completely set by the office's setup. The `AppID` key under which the security settings are recorded is not set. This poses no problem because the *dcomcnfg.exe* configuration tools sets it automatically.

To access the service manager remotely, the client must have launch and access permission. Those permissions appear as sub-keys of the `AppID` and have binary values. The values can be edited with dcomcnfg. Also the identity of the service manager must be set to “Interactive User”. When the office is started as a result of a remote activation of the service manager, it runs under the account of the currently logged-on user (the interactive user).

In case of callbacks (office calls into the client), the client must adjust its security settings so that incoming calls from the office are accepted. This happens when listener objects that are implemented as Automation objects (not UNO components) are passed as parameters to UNO objects, which in turn calls on those objects. Callbacks can also originate from the automation bridge, for example, when JScript `Array` objects are used. Then, the bridge modifies the `Array` object by its `IDispatchEx` interface. To get the interface, the bridge has to call `QueryInterface` with a call back to the client.

To avoid these callbacks, `VBArray` objects and `Value Objects` could be used.

To set security properties on a client, use the security API within a client program or make use of *dcomcnfg* again. The API can be difficult to use. Modifying the registry is the easiest method, simplified by *dcomcnfg*. This also adds more flexibility, because administrators can easily change the settings without editing source code and rebuilding the client. However, *dcomcnfg* only works with COM servers and not with ordinary executables. To use *dcomcnfg*, put the client code into a server that can be registered on the client machine. This not only works with exe servers, but also with in-process servers, namely dlls. Those can have an `AppID` entry when they are remote, that is, they have the `DllSurrogate` subkey set. To activate them an additional executable which instantiates the in-process server is required. At the first call on an interface of the server DCOM initializes security by using the values from the registry, but it only works if the executable has not called `CoInitializeSecurity` beforehand.

To run JScript or VBScript programs, an additional program, a script controller that runs the script is required, for example, the *Windows Scripting Host* (WSH). The problem with these controllers is that they might impose their own security settings by calling `CoInitializeSecurity` on their own behalf. In that case, the security settings that were previously set for the controller in the registry are not being used. Also, the controller does not have to be configurable by dcomcnfg, because it might not be a COM server. This is the case with WSH (not WSH remote).

To overcome these restrictions write a script controller that applies the security settings before a scripting engine has been created. This is time consuming and requires some knowledge about the engine, along with good programming skills. The *Windows Script Components* (WSC) is easier to use. A WSC is made of a file that contains XML, and existing JScript and VBS scripts can be put

into the respective XML Element. A wizard generates it for you. The WSC must be registered, which can be done with *regsvr32.exe* or directly through the context menu in the file explorer. To have an `AppID` entry, declare the component as remotely accessible. This is done by inserting the `remotable` attribute into the registration element in the wsc file:

```
<registration
    description="writerdemo script component"
    progid="dcomtest.writerdemo.WSC"
    version="1.00"
    classid="{90c5ca1a-5e38-4c6d-9634-b0c740c569ad}"
    remotable="true">
```

When the WSC is registered, there will be an appropriate `AppID` key in the registry. Use dcomcnfg to apply the desired security settings on this component. To run the script. An executable is required. For example:

```
Option Explicit
Sub main()
    Dim obj As Object
    Set obj = CreateObject("dcomtest.writerdemo.wsc")
    obj.run
End Sub
```

In this example, the script code is contained in the run function. This is how the wsc file appears:

```
<?xml version="1.0"?>
<component>
<?component error="true" debug="true"?>
<registration
    description="writerdemo script component"
    progid="dcomtest.writerdemo.WSC"
    version="1.00"
    classid="{90c5ca1a-5e38-4c6d-9634-b0c740c569ad}"
    remotable="true">
</registration>
<public>
    <method name="run">
    </method>
</public>
<script language="JScript">
<![CDATA[
var description = new jscripttest;
function jscripttest()
{
    this.run = run;
}
function run()
{
var objServiceManager= new ActiveXObject("com.sun.star.ServiceManager","\\jl-1036");
var objCoreReflection= objServiceManager.createInstance("com.sun.star.reflection.CoreReflection");
var objDesktop= objServiceManager.createInstance("com.sun.star.frame.Desktop");
var objCoreReflection= objServiceManager.createInstance("com.sun.star.reflection.CoreReflection");
var args= new Array();
var objDocument= objDesktop.loadComponentFromURL("private:factory/swriter", "_blank", 0, args);
var objText= objDocument.getText();
var objCursor= objText.createTextCursor();
objText.insertString( objCursor, "The first line in the newly created text document.\n", false);
objText.insertString( objCursor, "Now we're in the second line", false);
var objTable= objDocument.createInstance( "com.sun.star.text.TextTable");objTable.initialize( 4, 4);
objText.insertTextContent( objCursor, objTable, false);
var objRows= objTable.getRows();
var objRow= objRows.getByIndex( 0);
objTable.setPropertyValue( "BackTransparent", false);
objTable.setPropertyValue( "BackColor", 13421823);
objRow.setPropertyValue( "BackTransparent", false);
objRow.setPropertyValue( "BackColor", 6710932);
insertIntoCell( "A1","FirstColumn", objTable);
insertIntoCell( "B1","SecondColumn", objTable);
insertIntoCell( "C1","ThirdColumn", objTable);
insertIntoCell( "D1","SUM", objTable);
objTable.getCellByName("A2").setValue( 22.5);
objTable.getCellByName("B2").setValue( 5615.3);
objTable.getCellByName("C2").setValue( -2315.7);
objTable.getCellByName("D2").setFormula("sum <A2:C2>");objTable.getCellByName("A3").setValue( 21.5);
objTable.getCellByName("B3").setValue( 615.3);
objTable.getCellByName("C3").setValue( -315.7);
objTable.getCellByName("D3").setFormula( "sum <A3:C3>");objTable.getCellByName("A4").setValue( 121.5);
objTable.getCellByName("B4").setValue( -615.3);
objTable.getCellByName("C4").setValue( 415.7);
objTable.getCellByName("D4").setFormula( "sum <A4:C4>");
objCursor.setPropertyValue( "CharColor", 255);
objCursor.setPropertyValue( "CharShadowed", true);
```

```
objText.insertControlCharacter( objCursor, 0 , false);
objText.insertString( objCursor, " This is a colored Text - blue with shadow\n", false);
objText.insertControlCharacter( objCursor, 0, false );
var objTextFrame= objDocument.createInstance("com.sun.star.text.TextFrame”);
var objSize= createStruct("com.sun.star.awt.Size");
objSize.Width= 15000;
objSize.Height= 400;
objTextFrame.setSize( objSize);
objTextFrame.setPropertyValue( "AnchorType", 1);
objText.insertTextContent( objCursor, objTextFrame, false);
var objFrameText= objTextFrame.getText();
var objFrameTextCursor= objFrameText.createTextCursor();
objFrameText.insertString( objFrameTextCursor, "The first line in the newly created text frame.",
                false);
objFrameText.insertString(objFrameTextCursor,
                          "With this second line the height of the frame raises.", false );
objFrameText.insertControlCharacter( objCursor, 0 , false);
objCursor.setPropertyValue( "CharColor", 65536);
objCursor.setPropertyValue( "CharShadowed", false);
objText.insertString( objCursor, " That's all for now !!", false );

function insertIntoCell( strCellName, strText, objTable)
{
    var objCellText= objTable.getCellByName( strCellName);
    var objCellCursor= objCellText.createTextCursor();
    objCellCursor.setPropertyValue( "CharColor",16777215);
    objCellText.insertString( objCellCursor, strText, false);
}
function createStruct( strTypeName)
{
    var classSize= objCoreReflection.forName( strTypeName);
    var aStruct= new Array();
    classSize.createObject( aStruct);
    return aStruct[0];
}
}
]]>
</script>
</component>
```

This WSC contains the WriterDemo example written in JScript.

The Bridge Services

Service: com.sun.star.bridge.OleBridgeSupplier2

The component implements the `com.sun.star.bridge.XBridgeSupplier2` interface and converts Automation values to UNO values. The mapping of types occurs according to the mappings defined in *3.4.4 Professional UNO - UNO Language Bindings - Automation Bridge - Type Mappings*.



Usually you do not use this service unless you must convert a type manually.

A programmer uses the `com.sun.star.ServiceManager` ActiveX component to access the office. The COM class factory for `com.sun.star.ServiceManager` uses `OleBridgeSupplier2` internally to convert the UNO service manager into an Automation object. Another use case for the OleBridgeSupplier2 might be to use the SDK without an office installation. For example, if there is a UNO component from COM, write code which converts the UNO component without the need of an office. That code could be placed into an ActiveX object that offers a function, such as `getUNOComponent()`.

The interface is declared as follows:

```
module com {  module sun {  module star {  module bridge {

interface XBridgeSupplier2: com::sun::star::uno::XInterface
{
    any createBridge( [in] any aModelDepObject,
                      [in] sequence< byte > aProcessId,
                      [in] short nSourceModelType,
                      [in] short nDestModelType )
```

```
        raises( com::sun::star::lang::IllegalArgumentException );

}; }; }; };
```

The value that is to be converted and the converted value itself are contained in anys. The any is similar to the VARIANT type in that it can contain all possible types of its type system, but that type system only comprises UNO types and not Automation types. However, it is necessary that the function is able to receive as well as to return Automation values. In C++, void pointers could have been used, but pointers are not used with UNO IDL. Therefore, the any can contain a pointer to a VARIANT and that the type should be an unsigned long.

To provide the any, write this C++ code:

```
Any automObject;
// pVariant is a VARIANT* and contains the value that is going to be converted
automObject.setValue((void*) &pVariant, getCppuType((sal_uInt32*)0));
```

Whether the argument aModelDepObject or the return value carries a VARIANT depends on the mode in which the function is used. The mode is determined by supplying constant values as the nSourceModelType and nDestModelType arguments. Those constant are defined as follows:

```
module com {  module sun {  module star {  module bridge {
constants ModelDependent
{
    const short UNO = 1;
    const short OLE = 2;
    const short JAVA = 3;
    const short CORBA = 4;
};
}; }; }; };
```

The table shows the two possible modes:

nSourceModelType	nDestModelType	aModelDepObject	Return Value
UNO	OLE	contains UNO value	contains VARIANT*
OLE	UNO	contains VARIANT*	contains UNO value

When the function returns a VARIANT*, that is, a UNO value is converted to an Automation value, then the caller has to free the memory of the VARIANT:

```
sal_uInt8 arId[16];
rtl_getGlobalProcessId( arId );
Sequence<sal_Int8> procId((sal_Int8*)arId, 16);
Any anyDisp= xSupplier->createBridge( anySource, procId, UNO, OLE);
IDispatch* pDisp;
if( anyDisp.getValueTypeClass() == TypeClass_UNSIGNED_LONG)
{
    VARIANT* pvar= *(VARIANT**)anyDisp.getValue();
    if( pvar->vt == VT_DISPATCH)
    {
        pDisp= pvar->pdispVal;
        pDisp->AddRef();
    }
    VariantClear( pvar);
    CoTaskMemFree( pvar);
}
```

The function also takes a process ID as an argument. The implementation compares the ID with the ID of the process the component is running in. Only if the IDs are identical a conversion is performed. Consider the following scenario:

There are two processes. One process, the server process, runs the OleBridgeSupplier2 service. The second, the client process, has obtained the XBridgeSupplier2 interface by means of the UNO remote bridge. In the client process an Automation object is to be converted and the function XBridgeSupplier2::createBridge is called. The interface is actually a UNO interface proxy and the remote bridge will ensure that the arguments are marshaled, sent to the server process and that the original interface is being called. The argument aModelDepObject contains an IDispatch* and must be marshaled as COM interface, but the remote bridge only

sees an any that contains an `unsigned long` and marshals it accordingly. When it arrives in the server process, the `IDispatch*` has become invalid and calls on it might crash the application.

Service: com.sun.star.bridge.OleBridgeSupplierVar1

This service is a variation of the `OleBridgeSupplier2` service. The functionality is the same, but the implementation is optimized for a deployment scenario where remote UNO objects are converted into Automation objects. The UNO object is only a proxy and the actual object resides in a different process or on a different machine. To get a proxy of a remote object, establish a connection to another process which can run on another machine by means of the respective UNO mechanisms. Refer to *3.3.1 Professional UNO - UNO Concepts - UNO Interprocess Connections* for additional information. Calls on the proxy object result in an interprocess call that may take a long time.

To call a function of an Automation object, a `DISPID` must be obtained first. The ID is obtained by calling `IDispatch::GetIDsOfNames`. The `GetIDsOfName` takes a function or property name as an argument and returns a `DISPID` that is used in the Invoke call. Automation objects created by `OleBridgeSupplier2` verify in their `GetIDsOfName` implementation if the function or property with the specified name exists, involving one or two calls to the UNO object the first time the object's `GetIDsOfName` function is called. `OleBridgeSupplierVar1` handles that differently. The first time an object is being asked for a `DISPID`, the ID is generated and returned without verifying if there is a member of that name. When `Invoke` is called with that `DISPID` and the call fails, the bridge repeats the call with a verified name. Also, `Invoke` is often called with a combination of the flag `DISPATCH_METHOD` and one of the property flag, signifying that the `DISPID` represents a certain function or property. In that case, the bridge first presumes that the ID represents a function and performs the call accordingly. If that fails, it tries to access a property with that name. When the call eventually succeeds, the acquired information (for example, the verified name of the member, property or function) is cached in case the call is repeated.

The `OleBridgeSupplier2` and `OleBridgeSupplierVar1` services use the `com.sun.star.script.Invocation` service to convert UNO objects to UNO objects that implement com.sun.star.script.Invocation. Then the `XInvocation` objects are converted into `IDispatch` objects. `OleBridgeSupplierVar1` can be passed a service manager as an argument during instantiation (`com.sun.star.lang.XMultiServiceFactory:createInstanceWithArguments()`). It will then use that service manager to create the invocation service. If the service manager happens to be the remote service manager (provided by the server, for example, a remote office), the Invocation service is created on the server-side. Hence, all conversions of UNO objects to `XInvocation` objects occur remotely on the server and do not cause excessive network traffic.

Service: com.sun.star.bridge.OleApplicationRegistration

This service registers a COM class factory when the service is being instantiated and deregisters it when the service is being destroyed. The class factory creates a service manager as an Automation object. All UNO objects created by the service manager are then automatically converted into Automation objects.

Service: com.sun.star.bridge.OleObjectFactory

This service creates ActiveX components and makes them available as UNO objects which implement `XInvocation`. For the purpose of component instantiation, the OleClient implements the `com.sun.star.lang.XMultiServiceFactory` interface. The COM component is specified by its programmatic identifier (ProgId).

Although any ActiveX component with a ProgId can be created, a component can only be used if it supports `IDispatch` and provides type information through `IDispatch::GetTypeInfo`.

Unsupported COM Features

The Automation objects provided by the bridge do not provide type information. That is, `IDispatch::GetTypeInfoCount` and `IDispatch::GetTypeInfo` return `E_NOTIMPL`. Also, there are no COM type libraries available and the objects do not implement the `IProvideClassInfo[2]` interface.

`GetIDsOfName` processes only one name at a time. If an array of names is passed, then a `DISPID` is returned for the first name.

`IDispatch::Invoke` does not support named arguments and the *pExcepInfo and puArgErr* parameter.

4 Writing UNO Components

OpenOffice.org can be extended by UNO components. UNO components are shared libraries or jar files with the ability to instantiate objects that can integrate themselves into the UNO environment. A UNO component can access existing features of OpenOffice.org, and it can be used from within OpenOffice.org through the object communication mechanisms provided by UNO.

OpenOffice.org provides many entry points for these extensions.

- Arbitrary objects written in Java or C++ can be called from the user interface, display their own GUI, and work with the entire application.
- Calc Add-Ins can be used to create new formula sets that are presented in the formula autopilot.
- Chart Add-Ins can insert new Chart types into the charting tool.
- New database drivers can be installed into the office to extend data access.
- Entire application modules are exchangeable, for instance the linguistics module.
- It is possible to create new document types and add them to the office. For instance, a personal information manager could add message, calendar, task and journal document components, or a project manager could support a new project document.
- Developers can leverage the OpenOffice.org XML file format to read and write new file formats through components.

From OpenOffice.org 1.1 there is comprehensive support for component extensions. The entire product cycle of a component is now covered:

The design and development of components has been made easier by adding wizards for components to the NetBeans IDE. They are described in the directory docs/DevStudioWizard of the SDK. There are wizards for general components, for Calc AddIns and for IDL files.

Components can integrate themselves into the user interface, using simple configuration files. You can add new menus, toolbar items, and help items for a component simply by editing XML configuration files.

Component deployment is performed by a package installer, which inserts new components and their user interface extensions into networked and single installations of OpenOffice.org. During the production phase the package installer makes it simple to maintain components, to introduce bug fixes and new versions of a component. When a packaged component is no longer needed, it can easily be removed. This way, OpenOffice.org keeps the promise of being open for modular extensions.

Last but not least, this is not the only way to add features to the office. Learning how to write components and how to use the OpenOffice.org API at the same time teaches you the techniques used in the OpenOffice.org code base, thus enabling you to work with the existing OpenOffice.org source code, extend it or introduce bug fixes.

Components are the basis for all of these extensions. This chapter teaches you how to write UNO components. It assumes that you have at least read the chapter *2 First Steps* and—depending on your target language—the section about the Java or C++ language binding in *3 Professional UNO*.

4.1 Required Files

OpenOffice.org Software Development Kit (SDK)

The SDK provides a build environment for your projects, separate from the OpenOffice.org build environment. It contains the necessary tools for UNO development, C and C++ libraries and include files, Java packages, UNO type definitions and example code. But most of the necessary libraries and Java UNO packages are shared with an existing OpenOffice.org installation which is a prerequisite for a SDK.

The SDK development tools (executables) contained in the SDK are used in the following chapter. Become familiar with the following table that lists the executables from the SDK. These executables are found in the platform specific bin folder of the SDK installation. In Windows, they are in the folder *<SDK>\windows\bin*, on Linux they are stored in *<SDK>/linux/bin* and on Solaris in *<SDK>/solaris/bin*.

Executable	Description
idlc	The UNOIDL compiler that creates binary type description files with the extension .urd for registry database files.
idlcpp	The idlc preprocessor used by idlc.
cppumaker	The C++ UNO maker that generates headers with UNO types mapped from binary type descriptions to C++ from binary type descriptions.
javamaker	Java maker that generates interface and class definitions for UNO types mapped from binary type descriptions to Java from binary type descriptions.
xml2cmp	XML to Component that can extract type names from XML object descriptions for use with cppumaker and javamaker, creates functions.
regmerge	The registry merge that merges binary type descriptions into registry files.
regcomp	The register component that tells a registry database file that there is a new component and where it can be found.
pkgchk	The package check that installs components into an installed OpenOffice.org.
regview	The registry view that outputs the content of a registry database file in readable format.
autodoc	The automatic documentation tool that evaluates Javadoc style comments in idl files and generates documentation from them.
rdbmaker	The registry database maker that creates registry files with selected types and their dependencies.
uno	The UNO executable. It is a standalone UNO environment which is able to run UNO components supporting the `com.sun.star.lang.XMain` interface, one possible use is: `$ uno -s ServiceName -r MyRegistry.rdb -- MyMainClass arg1`

GNU Make

The makefiles in the SDK assume that the GNU *make* is used. Documentation for GNU *make* command line options and syntax are available at www.gnu.org. In Windows, not every GNU make seems stable, notably some versions of Cygwin *make* were reported to have problems with the SDK makefiles. Other GNU *make* binaries, such as the one from

unixutils.sourceforge.net work well even on the Windows command line. The package UnxUtils comes with a *zsh* shell and numerous utilities, such as *find*, *sed*. To install UnxUtils, download and unpack the archive, and add *<UnxUtils>\usr\local\wbin* to the PATH environment variable. Now launch *sh.exe* from *<UnxUtils>\bin* and issue the command *make* from within *zsh* or use the Windows command line to run *make*. For further information about *zsh*, go to *zsh.sunsite.dk*.

4.2 Using UNOIDL to Specify New Components

Component development does not necessarily start with the declaration of new interfaces or new types. Try to use the interfaces and types already defined in the OpenOffice.org API. If existing interfaces cover your requirements and you need to know how to implement them in your own component, go to section *4.3 Writing UNO Components - Component Architecture*. The following describes how to declare your own interfaces and other types you might need.

UNO uses its own meta language *UNOIDL* (UNO Interface Definition Language) to specify types. Using a meta language for this purpose enables you to generate language specific code, such as header files and class definitions, to implement objects in any target language supported by UNO. UNOIDL keeps the foundations of UNO language independent and takes the burden of mechanic language adaptation from the developer's shoulders when implementing UNO objects.

To define a new interface, service or other compound type, write its specification in UNOIDL, then compile it with the UNOIDL compiler *idlc*. After compilation, merge the resulting binary type description into a registry database that is used by *cppumaker* and *javamaker* during the make process to create necessary header and class files, and used by UNO during runtime to provide runtime type information. The chapter *3 Professional UNO* provides the various type mappings used by cppumaker and javamaker in the language binding sections. Refer to the section *4.9.2 Writing UNO Components - Deployment Options for Components - Background: UNO Registries - UNO Type Library* for details about type information in the registry database..



When writing your own specifications, please consult the chapter *A IDL Design Guide* which treats design principles and conventions used in API specifications. Follow the rules for universality, orthogonality, inheritance and uniformity of the API as described in the Design Guide.

4.2.1 Writing the Specification

There are similarities between C++, CORBA IDL and UNOIDL, especially concerning the syntax and the general usage of the compiler. If you are familiar with reading C++ or CORBA IDL, you will be able to understand much of UNOIDL, as well.

As a first example, consider the IDL specification for the `com.sun.star.bridge.XUnoUrlResolver` interface. An idl file usually starts with a number of preprocessor directives, followed by module instructions and a type definition:

```
#ifndef __com_sun_star_bridge_XUnoUrlResolver_idl__
#define __com_sun_star_bridge_XUnoUrlResolver_idl__

#include <com/sun/star/uno/XInterface.idl>
#include <com/sun/star/lang/IllegalArgumentException.idl>
#include <com/sun/star/connection/ConnectionSetupException.idl>
#include <com/sun/star/connection/NoConnectException.idl>


module com {  module sun {  module star {  module bridge {


/** service <type scope="com::sun::star::bridge">UnoUrlResolver</type>
```

```
        implements this interface.
 */
interface XUnoUrlResolver: com::sun::star::uno::XInterface
{
    // method com::sun::star::bridge::XUnoUrlResolver::resolve
    /** resolves an object, on the UNO URL.
     */
    com::sun::star::uno::XInterface resolve( [in] string sUnoUrl )
        raises (com::sun::star::connection::NoConnectException,
                com::sun::star::connection::ConnectionSetupException,
                com::sun::star::lang::IllegalArgumentException);
};

}; }; }; };

#endif
```

We will discuss this idl file step by step below, and we will write our own UNOIDL specification as soon as possible. The file specifying `com.sun.star.bridge.XUnoUrlResolver` is located in the *idl* folder of your SDK installation, *<SDK>/idl/com/sun/star/bridge/XUnoUrlResolver.idl.*

UNOIDL definition file names have the extension *.idl* by convention. The descriptions must use the US ASCII character set without special characters and separate symbols by whitespace, i.e. blanks, tabs or linefeeds.

Preprocessing

Just like a C++ compiler, the UNOIDL compiler *idlc* can only use types it already knows. The idlc knows 15 fundamental types such as boolean, int or string (they are summarized below). Whenever a type other than a fundamental type is used in the idl file, its declaration has to be included first. For instance, to derive an interface from the interface `XInterface`, include the corresponding file `XInterface.idl`. Including means telling the preprocessor to read a given file and execute the instructions found in it.

```
#include <com/sun/star/uno/XInterface.idl> // searched in include path given in -I parameter
#include "com/sun/star/uno/XInterface.idl" // searched in current path, then in include path
```

There are two ways to include idl files. A file name in angled brackets is searched on the include path passed to *idlc* using its -I option. File names in double quotes are first searched on the current path and then on the include path.

The `XUnoUrlResolver` definition above includes `com.sun.star.uno.XInterface` and the three exceptions thrown by the method `resolve()`, `com.sun.star.lang.IllegalArgumentException`, `com.sun.star.connection.ConnectionSetupException` and `com.sun.star.connection.NoConnectException`.

Furthermore, to avoid warnings about redefinition of already included types, use `#ifndef` and `#define` as shown above. Note how the entire definition for `XUnoUrlResolver` is enclosed between `#ifndef` and `#endif`. The first thing the preprocessor does is to check if the flag `__com_sun_star_bridge_XUnoUrlResolver_idl__` has already been defined. If not, the flag is defined and *idlc* continues with the definition of `XUnoUrlResolver`.

Adhere to the naming scheme for include flags used by the OpenOffice.org developers: Use the file name of the IDL file that is to be included, add double underscores at the beginning and end of the macro, and replace all slashes and dots by underscores.

For other preprocessing instructions supported by *idlc* refer to Bjarne Stroustrup: The C++ Programming Language.

Grouping Definitions in Modules

To avoid name clashes and allow for a better API structure, UNOIDL supports naming scopes. The corresponding instruction is `module`:

```
module mymodule {
};
```

Instructions are only known inside the module `mymodule` for every type defined within the pair of braces of this `module {}`. Within each module, the type identifiers are unique. This makes an UNOIDL module similar to a Java package or a C++ namespace.

Modules may be nested. The following code shows the interface `XUnoUrlResolver` contained in the module `bridge` that is contained in the module `star`, which is in turn contained in the module `sun` of the module `com`.

```
module com {  module sun {  module star {  module bridge {
    // interface XUnoUrlResolver in module com::sun::star::bridge
}; }; }; };
```

It is customary to write module names in lower case letters. Use your own module hierarchy for your IDL types. To contribute code to OpenOffice.org, use the `org::openoffice` namespace or `com::sun::star`. Discuss the name choice with the leader of the API project on *www.openoffice.org* to add to the latter modules. The `com::sun::star` namespace mirrors the historical roots of OpenOffice.org in StarOffice and will probably be kept for compatibility purposes.

Types defined in UNOIDL modules have to be referenced using full-type or scoped names, that is, you must enter all modules your type is contained in and separate the modules by the scope operator `::`. For instance, to reference `XUnoUrlResolver` in another idl definition file, write `com::sun::star::bridge::XUnoUrlResolver`.

Besides, modules have an advantage when it comes to generating language specific files. The tools *cppumaker* and *javamaker* automatically create subdirectories for every referenced `module`, if required. Headers and class definitions are kept in their own folders without any further effort.

Fundamental Types

Before we can go about defining our first interface, you need to know the fundamental types you may use in your interface definition. You should already be familiar with the fundamental UNO types from the chapters *2 First Steps* and *3 Professional UNO*. Since we have to use them in idl definition files, we repeat the type keywords and their meaning here.

Fundamental UNO type	Type description
`char`	16-bit unicode character type
`boolean`	boolean type; true and false
`byte`	8-bit ordinal integer type
`short`	signed 16-bit ordinal integer type
`unsigned short`	unsigned 16-bit ordinal integer type
`long`	signed 32-bit ordinal integer type
`unsigned long`	unsigned 32-bit integer type
`hyper`	signed 64-bit ordinal integer type
`unsigned hyper`	unsigned 64-bit ordinal integer type
`float`	processor dependent float
`double`	processor dependent double
`string`	string of 16-bit unicode characters
`any`	universal type, takes every fundamental or compound UNO type, similar to Variant in other environments or Object in Java
`void`	Indicates that a method does not provide a return value

Defining an Interface

Interfaces describe aspects of objects. To specify a new behavior for the component, start with an interface definition that comprises the methods offering the new behavior. Define a pair of plain get and set methods in a single step using the `attribute` instruction. Alternatively, choose to define your own *operations* with arbitrary arguments and exceptions by writing the operation signature, and the exceptions the operation throws. We will first write a small interface definition with `attribute` instructions, then consider the `resolve()` operation in `XUNoUrlResolver`.

Let us assume we want to contribute an `ImageShrink` component to OpenOffice.org to create thumbnail images for use in OpenOffice.org tables. There is already a `com.sun.star.document.XFilter` Interface offering methods supporting file conversion. In addition, a method is required to get and set the source and target directories, and the size of the thumbnails to create. It is common practice that a service and its prime interface have corresponding names, so our component shall have an `org::openoffice::test::XImageShrink` interface with methods to do so through get and set methods.

Attributes

The `attribute` instruction creates these methods for the experimental interface definition:

Look at the specification for our `XImageShrink` interface[1]:
(Components/Thumbs/org/openoffice/test/XImageShrink.idl)

```
#ifndef __org_openoffice_test_XImageShrink_idl__
#define __org_openoffice_test_XImageShrink_idl__
#include <com/sun/star/uno/XInterface.idl>
#include <com/sun/star/awt/Size.idl>
```

1 Perhaps in real life it would be better to define a more universal XBatchConverter interface for the source and target directories and derive XImageShrink from it. There are other options as well, but we want to keep things simple.

```
module org { module openoffice { module test {

interface XImageShrink : com::sun::star::uno::XInterface
{
    [attribute] string SourceDirectory;
    [attribute] string DestinationDirectory;
    [attribute] com::sun::star::awt::Size Dimension;
};

}; }; };

#endif
```



OpenOffice.org API interfaces do not use attributes anymore, because it entices programmers into ignoring exceptions. They are confusing, because attributes are mapped as prefixed get/set methods in an implementation language like Java or C++. It is sometimes difficult to match these methods with the original attribute declaration. Also note, that attribute definitions in UNOIDL interfaces do not declare any data fields, just the access methods.

We protect the interface from being redefined using `#ifndef`, then added `#include com.sun.star.uno.XInterface` and the struct `com.sun.star.awt.Size`. These were found in the API reference using its global index. Our interface will be known in the `org::openoffice::test` module, so it is nested in the corresponding `module` instructions.

Define an interface using the `interface` instruction. It opens with the keyword `interface`, gives an interface name and derives the new interface from a parent interface (also called super interface). It then defines the interface body in braces. The `interface` instruction concludes with a semicolon.

In this case, the introduced interface is `XImageShrink`. By convention, all interface identifiers start with an X. Every interface must inherit from the base interface for all UNO interfaces `XInterface` or from one of its derived interfaces. UNO supports single inheritance, so you may only inherit from *one* interface. Inheritance is expressed by a colon `:` followed by the *fully qualified name* of the parent type. The fully qualified name of a UNOIDL type is its identifier, including all containing modules separated by the scope operator `::`. Here we derive from `com::sun::star::uno::XInterface` directly.



UNOIDL allows forward declaration of interfaces used as parameters, return values or struct members. However, an interface you want to derive from must be a fully defined interface.

After the super interface the interface body begins. It may contain attribute instructions or operations. Consider the interface body of `XImageShrink`. It contains three attributes and no operation. The operations are discussed below.

An `attribute` instruction opens with the keyword `attribute` in square brackets, then it gives a known type and an identifier for the attribute, and concludes with a semicolon.

In our example, the `string` attributes named `SourceDirectory` and `DestinationDirectory` and a `com::sun::star::awt::Size` attribute known as `Dimension` were defined:

```
    [attribute] string SourceDirectory;
    [attribute] string DestinationDirectory;
    [attribute] com::sun::star::awt::Size Dimension;
```

During code generation, the `attribute` instruction leads to pairs of get and set methods. For instance, the Java interface generated by *javamaker* from this type description contains the following six methods. Note that no exceptions can be specified for attribute methods:

```
    // from attribute SourceDir
    public String getSourceDirectory();
    public void setSourceDirectory(String _sourcedir);

    // from attribute DestinationDir
    public String getDestinationDirectory();
    public void setDestinationDirectory(String _destinationdir);

    // from attribute Dimension
```

```
    public com.sun.star.awt.Size getDimension();
    public void setDimension(com.sun.star.awt.Size _dimension);
```

As an option, define that an attribute cannot be changed from the outside using a `readonly` flag. To set this flag, write `[attribute, readonly]`. The effect is that only a `get()` method is created during code generation, but not a `set()` method.

Operations

When writing a real component, define the *operations* by providing their signature and the exceptions they throw in the idl file. Our `XUnoUrlResolver` example above features a `resolve()` operation taking a UNO URL and throwing three exceptions.

```
interface XUnoUrlResolver: com::sun::star::uno::XInterface
{
    com::sun::star::uno::XInterface resolve( [in] string sUnoUrl )
        raises (com::sun::star::connection::NoConnectException,
                com::sun::star::connection::ConnectionSetupException,
                com::sun::star::lang::IllegalArgumentException);
};
```

The basic structure of an operation is similar to C++ functions or Java methods. The operation is defined giving a known return type, the operation name, an argument list in brackets `()` and if necessary, a list of the exceptions the operation may throw. The argument list, the exception clause `raises()` and an optional `[oneway]` flag preceding the operation are special in UNOIDL.

- Each argument in the argument list must commence with one of the direction flags `[in]`, `[out]` or `[inout]` before a known type and identifier for the argument is given. The direction flag specifies how the operation may use the argument:

Direction Flags for Operations	Description
`in`	Specifies that the operation shall evaluate the argument as input parameter, but it cannot change it.
`out`	Specifies that the argument does not parameterize the operation, instead the operation uses the argument as output parameter.
`inout`	Specifies that the operation is parameterized by the argument and that the operation uses the argument as output parameter as well.

  Avoid the [`inout`] and [`out`] qualifier. OpenOffice.org API interfaces do not use this qualifier.

- Exceptions are given through an optional `raises()` clause containing a comma-separated list of known exceptions given by their full name. The presence of a `raises()` clause means that only the listed exceptions, `com.sun.star.uno.RuntimeException` and their descendants may be thrown by the implementation. By specifying exceptions for operations, the implementer of your interface can return information to the caller, thus avoiding possible error conditions.
- If you prepend a `[oneway]` flag to an operation, the operation must perform its task asynchronously, that is, it should spawn a thread and return immediately. The argument list may be empty. Multiple arguments must be separated by commas. A `oneway` operation can not have a return value, or out or inout parameters.



You may not override an attribute or an operation inherited from a parent interface, that would not make sense in an abstract specification anyway. Furthermore, overloading is not possible. The qualified interface identifier in conjunction with the name of the method creates a unique method name.

Defining a Service

UNOIDL Services combine interfaces and properties to specify a certain functionality. In addition, services can include other services. For these purposes, the instructions `interface`, `property` and `service` are used within service specifications. Usually services are the basis for an object implementation, although there are services in the OpenOffice.org API that only serve as foundation or addition to other services, but are not meant to be implemented by themselves[2].

We are ready to assemble our `ImageShrink` service. Our service will read image files from a source directory and write shrinked versions of the found images to a destination directory. Our `XImageShrink` interface offers the needed capabilities, together with the interface `com.sun.star.document.XFilter` that supports two methods:

```
boolean filter( [in] sequence< com::sun::star::beans::PropertyValue > aDescriptor)
void cancel()
```

The following code shows the `ImageShrink` service specification: (Components/Thumbs/org/openoffice/test/ImageShrink.idl)

```
#ifndef __org_openoffice_test_ImageShrink_idl__
#define __org_openoffice_test_ImageShrink_idl__
#include <org/openoffice/test/XImageShrink.idl>

module org { module openoffice { module test {

service ImageShrink
{
    interface org::openoffice::test::XImageShrink;
    interface com::sun::star::document::XFilter;
};

}; }; };

#endif
```

Define a service using the `service` instruction. It opens with the keyword `service`, followed by a service name and the service body in braces. The `service` instruction concludes with a semicolon. Here we defined a service `ImageShrink`. The first letter of a service name should be an upper-case letter. The body of a service can reference interfaces and services using `interface` and `service` instructions, and it can identify properties supported by the service through `[property]` instructions.

- `interface` instructions followed by interface names in a service body indicates that the service supports these interfaces. By default, the `interface` forces the developer to implement this interface. To suggest an interface for a certain service, prepend an `[optional]` flag in front of the keyword `interface`. This weakens the specification to a permission. An optional interface can be implemented. Use one `interface` instruction for each supported interface or give a comma-separated list of interfaces to be exported by a service. You must terminate the `interface` instruction using a semicolon.
- `service` instructions in a service body include other services. The effect is that all interface and property definitions of the other services become part of the current service. A service reference can be optional using the `[optional]` flag in front of the `service` keyword. Use one instruction per service or a comma-separated list for the services to reference. The `service` instruction ends with a semicolon.

2 The services `com.sun.star.text.BaseFrame` or `com.sun.star.style.CharacterProperties` are part of other services, but are not implemented as such anywhere.

- `[property]` instructions describe qualities of a service that can be reached from the outside under a particular name and type. As opposed to interface attributes, these qualities are not considered to be a structural part of a service. Refer to the section *3.3.4 Professional UNO - UNO Concepts - Properties* in the chapter *3 Professional UNO* to determine when to use interface attributes and when to introduce properties in a service . The `property` instruction must be enclosed in square brackets, and continue with a known type and a property identifier. Just like a service and an interface, make a property non-mandatory writing `[property, optional]`. Besides `optional`, there is a number of other flags to use with properties. The following table shows all flags that can be used with `[property]`:

Property Flags	Description
`optional`	Property is non-mandatory.
`readonly`	The value of the property cannot be changed using the setter methods for properties, such as `setPropertyValue(string name)`.
`bound`	Changes of values are broadcast to `com.sun.star.beans.XPropertyChangeListeners` registered with the component.
`constrained`	The component must broadcast an event before a value changes, listeners can veto.
`maybeambiguous`	The value cannot be determined in some cases, for example, in multiple selections.
`maybedefault`	The value might come from a style or the application environment instead of from the object itself.
`maybevoid`	The property type determines the range of possible values, but sometimes there may be situations where there is no information available. Instead of defining special values for each type denoting that there are no meaningful values, the UNO type `void` can be used. Its meaning is comparable to `null` in relational databases.
`removable`	The property is removable. If a property is made removable, you must check for the existence of a property using `hasPropertyByName()` at the interface `com.sun.star.beans.XPropertySetInfo` and consider providing the capability to add or remove properties using `com.sun.star.beans.XPropertyContainer`.
`transient`	The property will not be stored if the object is serialized (made persistent).

Several properties of the same type can be listed in one `property` instruction. Remember to add a semicolon at the end of the instruction. Implement the interface `com.sun.star.beans.XPropertySet` when putting properties in your service, otherwise the properties specified will not work for others using the component.



Some services, which specify no interfaces at all, only properties, are used as a sequence of `com.sun.star.beans.PropertyValue` in OpenOffice.org, for example, `com.sun.star.document.MediaDescriptor`.

The following UNOIDL snippet shows the service, the interfaces and the properties supported by the service `com.sun.star.text.TextDocument` as defined in UNOIDL. Note the optional interfaces and the optional and read-only properties.

```
service TextDocument
{
    service com::sun::star::document::OfficeDocument;

    interface com::sun::star::text::XTextDocument;
    interface com::sun::star::util::XSearchable;
    interface com::sun::star::util::XRefreshable;
    interface com::sun::star::util::XNumberFormatsSupplier;
```

```
    [optional] interface com::sun::star::text::XFootnotesSupplier;
    [optional] interface com::sun::star::text::XEndnotesSupplier;
    [optional] interface com::sun::star::util::XReplaceable;
    [optional] interface com::sun::star::text::XPagePrintable;
    [optional] interface com::sun::star::text::XReferenceMarksSupplier;
    [optional] interface com::sun::star::text::XLineNumberingSupplier;
    [optional] interface com::sun::star::text::XChapterNumberingSupplier;
    [optional] interface com::sun::star::beans::XPropertySet;
    [optional] interface com::sun::star::text::XTextGraphicObjectsSupplier;
    [optional] interface com::sun::star::text::XTextEmbeddedObjectsSupplier;
    [optional] interface com::sun::star::text::XTextTablesSupplier;
    [optional] interface com::sun::star::style::XStyleFamiliesSupplier;

    [optional, property] com::sun::star::lang::Locale CharLocale;
    [optional, property] string WordSeparator;

    [optional, readonly, property] long CharacterCount;
    [optional, readonly, property] long ParagraphCount;
    [optional, readonly, property] long WordCount;

};
```



You might encounter two more instructions in service bodies. The instruction `observes` can stand in front of interface references and means that the given interfaces must be "observed". Since the `observes` instruction is disapproved of, no further explanation is provided.
If a service references another service using the keyword `needs` in front of the reference, then this service depends on the availability of the needed service at runtime. Newly specified services should not use `needs` as it is considered too implementation specific.

Defining a Sequence

A sequence in UNOIDL is an array containing a variable number of elements of the same UNOIDL type. The following is an example of a `sequence` term:

```
// this term could occur in a UNOIDL definition block somewhere
sequence< com::sun::star::uno::XInterface >
```

It starts with the keyword `sequence` and gives the element type enclosed in angle brackets <>. The element type must be a known type. A sequence type can be used as parameter, return value, property or struct member just like any other type. Sequences can also be nested, if necessary.

```
// this could be a nested sequence definition
sequence< sequence< long > >

// this could be an operation using sequences in some interface definition
sequence< string > getNamesOfIndex(sequence< long > indexes);
```

Defining a Struct

A `struct` is a compound type which puts together arbitrary UNOIDL types to form a new data type. Its member data are not encapsulated, rather they are publicly available. Structs are frequently used to handle related data easily, and the event structs broadcast to event listeners.

A struct instruction opens with the keyword `struct`, gives an identifier for the new struct type and has a struct body in braces. It is terminated by a semicolon. The struct body contains a list of struct member declarations that are defined by a known type and an identifier for the struct member. The member declarations must end with a semicolon, as well.

```
#ifndef __com_sun_star_reflection_ParamInfo_idl__
#define __com_sun_star_reflection_ParamInfo_idl__

#include <com/sun/star/reflection/ParamMode.idl>

module com {  module sun {  module star {  module reflection {

interface XIdlClass; // forward interface declaration

struct ParamInfo
{
```

```
    string aName;
    ParamMode aMode;
    XIdlClass aType;
};

}; }; }; };

#endif
```

UNOIDL supports inheritance of `struct` types. Inheritance is expressed by a colon `:` followed by the *full name* of the parent type. A struct type recursively inherits all members of the parent `struct` and their parents. For instance, derive from the struct `com.sun.star.lang.EventObject` to put additional information about new events into customized event objects to send to event listeners.

```
// com.sun.star.beans.PropertyChangeEvent inherits from com.sun.star.lang.EventObject
// and adds property-related information to the event object
struct PropertyChangeEvent : com::sun::star::lang::EventObject
{
    string PropertyName;
    boolean Further;
    long PropertyHandle;
    any OldValue;
    any NewValue;
};
```

Defining an Exception

An `exception` type is a type that contains information about an error. If an operation detects an error that halts the normal process flow, it must raise an exception and send information about the error back to the caller through an `exception` object. This causes the caller to interrupt its normal program flow as well and react according to the information received in the exception object. For details about exceptions and their implementation, refer to the chapters *3.4 Professional UNO - UNO Language Bindings* and *3.3.6 Professional UNO - UNO Concepts - Exception Handling*.

There are a number of exceptions to use. The exceptions should be sufficient in many cases, because a message string can be sent back to the caller. When defining an exception, do it in such a way that other developers could reuse it in their contexts.

An exception instruction opens with the keyword `exception`, gives an identifier for the new exception type and has an exception body in braces. It is terminated by a semicolon. The exception body contains a list of exception member declarations that are defined by a known type and an identifier for the exception member. The member declarations must end with a semicolon, as well.

Exceptions must be based on `com.sun.star.uno.Exception` or `com.sun.star.uno.RuntimeException`, directly or indirectly through derived exceptions of these two exceptions. `com.sun.star.uno.Exceptions` can only be thrown in operations specified to raise them while `com.sun.star.uno.RuntimeExceptions` can always occur. Inheritance is expressed by a colon `:`, followed by the *full name* of the parent type.

```
// com.sun.star.uno.Exception is the base exception for all exceptions
exception Exception {
    string Message;
    XInterface Context;
};

// com.sun.star.lang.IllegalArgumentException tells the caller which
// argument caused trouble
exception IllegalArgumentException: com::sun::star::uno::Exception
{
    /** identifies the position of the illegal argument.
        <p>This field is -1 if the position is not known.</p>
     */
    short ArgumentPosition;

};

// com.sun.star.uno.RuntimeException is the base exception for serious errors
// usually caused by programming errors or problems with the runtime environment
exception RuntimeException : com::sun::star::uno::Exception {
};
```

```
// com.sun.star.uno.SecurityException is a more specific RuntimeException
exception SecurityException : com::sun::star::uno::RuntimeException {
};
```

Predefining Values

Predefined values can be provided, so that implementers do not have to use cryptic numbers or other literal values. There are two kinds of predefined values, constants and enums. Constants can contain values of any fundamental UNOIDL type, except string. The enums are automatically numbered `long` values.

Const and Constants

The `constants` type is a container for `const` types. A `constants` instruction opens with the keyword `constants`, gives an identifier for the new group of `const` values and has the body in braces. It terminates with a semicolon. The `constants` body contains a list of `const` definitions that define the values of the members starting with the keyword `const` followed by a known type name and the identifier for the `const` in uppercase letters. Each `const` definition must assign a value to the `const` using an equals sign. The value must match the given type and can be an integer or floating point number, or a character, or a suitable `const` value or an arithmetic term based on the operators in the table below. The `const` definitions must end with a semicolon, as well.

```
#ifndef __com_sun_star_awt_FontWeight_idl__
#define __com_sun_star_awt_FontWeight_idl__

module com {  module sun {  module star {  module awt {

constants FontWeight
{
    const float DONTKNOW = 0.000000;
    const float THIN = 50.000000;
    const float ULTRALIGHT = 60.000000;
    const float LIGHT = 75.000000;
    const float SEMILIGHT = 90.000000;
    const float NORMAL = 100.000000;
    const float SEMIBOLD = 110.000000;
    const float BOLD = 150.000000;
    const float ULTRABOLD = 175.000000;
    const float BLACK = 200.000000;
};

}; }; }; };
```

Operators Allowed in `const`	**Meaning**
+	addition
-	subtraction
*	multiplication
/	division
%	modulo division
-	negative sign
+	positive sign
|	bitwise or
^	bitwise xor
&	bitwise and
~	bitwise not
>> <<	bitwise shift right, shift left



Use `constants` to group `const` types. In the Java language, binding a `constants` group leads to one class for all `const` members, whereas a single `const` is mapped to an entire class.

Enum

An `enum` type holds a group of predefined `long` values and maps them to meaningful symbols. It is equivalent to the enumeration type in C++. An `enum` instruction opens with the keyword `enum`, gives an identifier for the new group of `enum` values and has an `enum` body in braces. It terminates with a semicolon. The `enum` body contains a comma-separated list of symbols in uppercase letters that are automatically mapped to `long` values counting from zero, by default.

```
#ifndef __com_sun_star_style_ParagraphAdjust_idl__
#define __com_sun_star_style_ParagraphAdjust_idl__

module com {  module sun {  module star {  module style {

enum ParagraphAdjust
{
    LEFT,
    RIGHT,
    BLOCK,
    CENTER,
    STRETCH
};

}; }; }; };
#endif
```

In this example, `com.sun.star.style.ParagraphAdjust:LEFT` corresponds to 0, ParagraphAdjust.RIGHT corresponds to 1 and so forth.

An `enum` member can also be set to a `long` value using the equals sign. All the following `enum` values are then incremented starting from this value. If there is another assignment later in the code, the counting starts with that assignment:

```
enum Error {
    SYSTEM = 10, // value 10
    RUNTIME,     // value 11
    FATAL,       // value 12
    USER = 30,   // value 30
    SOFT         // value 31
};
```



The explicit use of `enum` values is deprecated and should not be used. It is a historical characteristic of the `enum` type but it makes not really sense and makes, for example language bindings unnecessarily complicated.

Using Comments

Comments are code sections ignored by *idlc*. In UNOIDL, use C++ style comments. A double slash `//` marks the rest of the line as comment. Text enclosed between `/*` and `*/` is a comment that may span over multiple lines.

```
service ImageShrink
{
    // the following lines define interfaces:
    interface org::openoffice::test::XImageShrink; // our home-grown interface
    interface com::sun::star::document::XFilter;

    /* we could reference other interfaces, services and properties here.
       However, the keywords uses and needs are deprecated
    */
};
```

Based on the above, there are documentation comments that are extracted when idl files are processed with *autodoc*, the UNOIDL documentation generator. Instead of writing `/*` or `//`to mark a plain comment, write `/**` or `///` to create a documentation comment.

```
/** Don't repeat asterisks within multiple line comments,
  * <- as shown here
  */

/// Don't write multiple line documentation comments using triple slashes,
/// since only this last line will make it into the documentation
```

Our `XUnoUrlResolver` sample idl file contains plain comments and documentation comments.

```
/** service <type scope="com::sun::star::bridge">UnoUrlResolver</type>
    implements this interface.
 */
interface XUnoUrlResolver: com::sun::star::uno::XInterface
{
    // method com::sun::star::bridge::XUnoUrlResolver::resolve
    /** resolves an object, on the UNO URL.
     */

    ...
}
```

Note the additional `<type/>` tag in the documentation comment pointing out that the service `UnoUrlResolver` implements the interface `XUnoUrlResolver`. This tag becomes a hyperlink in HTML documentation generated from this file. The chapter *B IDL Documentation Guide* provides a comprehensive description for UNOIDL documentation comments.

Singleton

A `singleton` instruction defines a global name for a service instance and determines that there can only be one instance of this `service` that must be reachable under this name. In the future, there will be the capability of retrieving the singleton instance from the component context using the name of the `singleton`. If the `singleton` has not been instantiated yet, the component context creates it. A singleton instruction looks like this:

```
singleton theServiceManager {
    service com::sun::star::lang::ServiceManager;
};
```

Reserved Types

There are types in UNOIDL which are reserved for future use. The *idlc* will refuse to compile the specifications if they are tried.

Array

The keyword `array` is reserved, but it cannot be used in UNOIDL. There will be sets containing a fixed number of elements, as opposed to sequences, that can have an arbitrary number of elements.

Union

There is also a reserved keyword for `union` types that cannot be used in UNOIDL. A `union` will look at a variable value from more than one perspective. For instance, a union for a long value is defined and this same value is accessed as a whole, or accessed by its high and low part separately through a union.

4.2.2 Generating Source Code from UNOIDL Definitions

The type description provided in .idl files is used in the subsequent process to create type information for the service manager and to generate header and class files. Processing the UNOIDL definitions is a three-step process.

1. Compile the *.idl* files using *idlc*. The result are *.urd* files (UNO reflection data) containing binary type descriptions.
2. Merge the *.urd* files into a registry database using *regmerge*. The registry database files have the extension *.rdb* (registry database). They contain binary data describing types in a tree-like structure starting with / as the root. The default key for type descriptions is the /UCR key (UNO core reflection).
3. Generate sources from registry files using *javamaker or cppumaker*. The tools *javamaker* and *cppumaker* map UNOIDL types to Java and C++ as described in the chapter *3.4 Professional UNO - UNO Language Bindings*. The registries used by these tools must contain all types to map to the programming language used, including all types referenced in the type descriptions. Therefore, javamaker and cppumaker need the registry that was merged, but the entire office registry as well. OpenOffice.org comes with a complete registry database providing all types used by UNO at runtime. The SDK uses the database (type library) of an existing OpenOffice.org installation.

The following shows the necessary commands to create Java class files and C++ headers from *.idl* files in a simple setup under Linux. We assume the jars from *<OFFICE_PROGRAM_PATH>/classes* have been added to your CLASSPATH, the SDK is installed in */home/sdk*, and */home/sdk/linux/bin* is in the PATH environment variable, so that the UNO tools can be run directly. The project folder is */home/sdk/Thumbs* and it contains the above *.idl* file *XImageShrink.idl*.

```
# make project folder the current directory
cd /home/sdk/Thumbs

# compile XImageShrink.idl using idlc
#   usage: idlc [-options] file_1.idl ... file_n.idl
#   -C adds complete type information including services
#   -I includepath tells idlc where to look for include files
#
# idlc writes the resulting urds to the current folder by default
idlc -C -I../idl XImageShrink.idl

# create registry database (.rdb) file from UNO registry data (.urd) using regmerge
#   usage: regmerge mergefile.rdb mergeKey regfile_1.urd ... regfile_n.urd
#   mergeKey entry in the tree-like rdb structure where types from .urd should be recorded, the tree
#   starts with the root / and UCR is the default key for type descriptions
#
# regmerge writes the rdb to the current folder by default
regmerge thumbs.rdb /UCR XImageShrink.urd

# generate Java source files for new types from rdb
#   -B base node to look for types, in this case UCR
#   -T type to generate Java files for
#   -nD do not generate sources for dependent types, they are available in the Java UNO jar files
#
# javamaker creates a directory tree for the output files according to
# the modules the given types were placed in. The tree is created in the current folder by default
javamaker -BUCR -Torg.openoffice.test.XImageShrink -nD <OFFICE_PROGRAM_PATH>/applicat.rdb thumbs.rdb

# generate C++ header files (hpp and hdl) for new types and their dependencies from rdb
#   -B base node to look for types, in this case UCR
#   -T type to generate Java files for
#
# cppumaker creates a directory tree for the output files according to
# the modules the given types were placed in. The tree is created in the current folder by default
cppumaker -BUCR -Torg.openoffice.test.XImageShrink <OFFICE_PROGRAM_PATH>/applicat.rdb thumbs.rdb

# compile Java class for new type
javac -g org/openoffice/test/XImageShrink.java
```

After issuing these commands you have a registry database *thumbs.rdb* and a Java class file *XImageShrink.class*. You can run *regview* against *thumbs.rdb to* see what regmerge has accomplished.

```
regview thumbs.rdb
```

The result for our interface `XImageShrink` looks like this:

```
Registry "file:///home/sdk/Thumbs/thumbs.rdb":

/
 / UCR
   / org
     / openoffice
       / test
         / XImageShrink
           Value: Type = RG_VALUETYPE_BINARY
                  Size = 316
                  Data = minor version: 0
                         major version: 1
                         type: 'interface'
                         uik: { 0x00000000-0x0000-0x0000-0x00000000-0x00000000 }

                         name: 'org/openoffice/test/XImageShrink'
                         super name: 'com/sun/star/uno/XInterface'
                         Doku: ""
                         IDL source file: "/home/sdk/Thumbs/XImageShrink.idl"
                         number of fields: 3
                         field #0:
                           name='SourceDirectory'
                           type='string'
                           access=READWRITE

                           Doku: ""
                           IDL source file: ""
                         field #1:
                           name='DestinationDirectory'
                           type='string'
                           access=READWRITE

                           Doku: ""
                           IDL source file: ""
                         field #2:
                           name='Dimension'
                           type='com/sun/star/awt/Size'
                           access=READWRITE

                           Doku: ""
                           IDL source file: ""
                         number of methods: 0
                         number of references: 0
```

Source generation can be fully automated with makefiles. For details, see the sections *4.5.9 Writing UNO Components - Simple Component in Java - Running and Debugging Java Components* and *4.6.10 Writing UNO Components - C++ Component - Building and Testing C++ Components* below. You are now ready to implement your own types and interfaces in a UNO component. The next section discusses the UNO core interfaces to implement in UNO components.

4.3 Component Architecture

UNO components are archive files or dynamic link libraries with the ability to instantiate objects which can integrate themselves into the UNO environment. For this purpose, components must contain certain static methods (Java) or export functions (C++) to be called by a UNO service manager. In the following, these methods are called component operations.

There must be a method to supply single-service factories for each object implemented in the component. Through this method, the service manager can get a single factory for a specific object and ask the factory to create the object contained in the component. Furthermore, there has to be a method which writes registration information about the component, which is used when a compo-

nent is registered with the service manager. In C++, an additional function is necessary that informs the component loader about the compiler used to build the component.

The component operations are always necessary in components and they are language specific. Later, when Java and C++ are discussed, we will show how to write them.

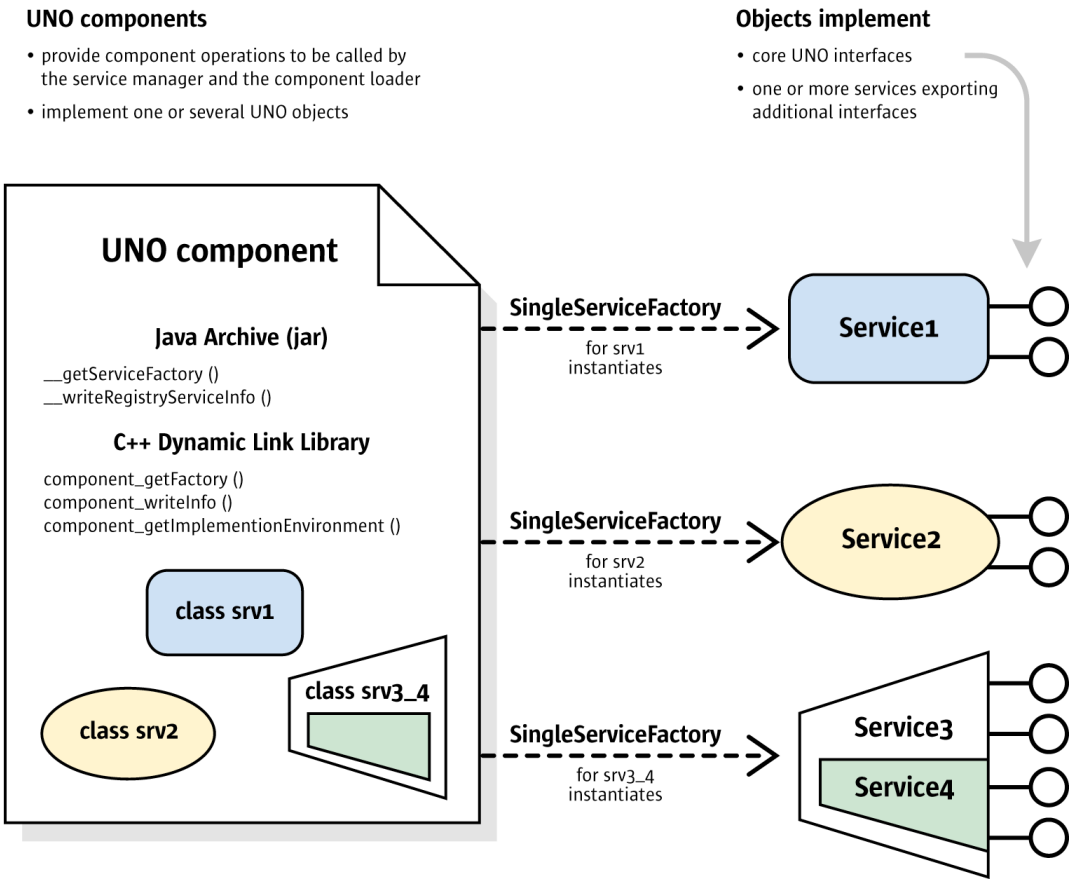


Illustration 34: A Component implementing three UNO objects

The illustration shows a component which contains three implemented objects. Two of them, srv1 and srv2 implement a single service specification (Service1 and Service2), whereas srv3_4 supports two services at once (Service3 and Service4).

The objects implemented in a component must support a number of core UNO interfaces to be fully usable from all parts of the OpenOffice.org application. These core interfaces are discussed in the next section. The individual functionality of the objects is covered by the additional interfaces they export. Usually these interfaces are enclosed in a service specification.

4.4 Core Interfaces to Implement

It is important to know where the interfaces to implement are located. The interfaces here are located at the object implementations in the component. When writing UNO components, the desired methods have to be implemented into the application and also, the core interfaces used to enable communication with the UNO environment. Some of them are mandatory, but there are others to choose from.

Interface	Required	Should be implemented	Optional	Special Cases	Helper class available for C++ and Java
XInterface	•				•
XTypeProvider		•			•
XServiceInfo		•			
XWeak		•			•
XComponent			•		•
XInitialization			•		
XMain				•	
XAggregation				•	
XUnoTunnel				•	

The interfaces listed in the table above have been characterized here briefly. More descriptions of each interface are provided later, as well as if helpers are available and which conditions apply.

com.sun.star.uno.XInterface

The component will not work without it. The base interface `XInterface` gives access to higher interfaces of the service and allows other objects to tell the service when it is no longer needed, so that it can destroy itself.

```
// com::sun::star::uno::XInterface

any queryInterface( [in] type aType );
[oneway] void acquire(); // increase reference counter in your service implementation
[oneway] void release(); // decrease reference counter, delete object when counter becomes zero
```

Usually developers do not call `acquire()` explicitly, because it is called automatically by the language bindings when a reference to a component is retrieved through `UnoRuntime.queryInterface()` or `Reference<destInterface>(sourceInterface, UNO_QUERY)`. The counterpart `release()` is called automatically when the reference goes out of scope in C++ or when the Java garbage collector throws away the object holding the reference.

com.sun.star.lang.XTypeProvider

This interface is used by scripting languages such as OpenOffice.org Basic to get type information. OpenOffice.org Basic cannot use the component without it.

```
// com::sun::star::lang::XTypeProvider

sequence<type> getTypes();
sequence<byte> getImplementationId();
```

com.sun.star.lang.XServiceInfo

This interface is used by other objects to get information about the service implementation.

```
// com::sun::star::lang::XServiceInfo

string getImplementationName();
boolean supportsService( [in] string ServiceName );
sequence<string> getSupportedServiceNames();
```

com.sun.star.uno.XWeak

This interface allows clients to keep a weak reference to the object. A weak reference does not prevent the object from being destroyed if another client keeps a hard reference to it, therefore it allows a hard reference to be retrieved again. The technique is used to avoid cyclic references. Even if the interface is not required by you, it could be implemented for a client that may want to establish a weak reference to an instance of your object.

```
// com.sun.star.uno.XWeak
```

```
com::sun::star::uno::XAdapter queryAdapter(); // creates Adapter
```

com.sun.star.lang.XComponent
This interface is used if cyclic references can occur in the component holding another object and the other object is holding a reference to that component. It can be specified in the service description who shall destroy the object.

```
// com::sun::star::lang::XComponent

void dispose(); //an object owning your component may order it to delete itself using dispose()
void addEventListener(com::sun::star::lang::XEventListener xListener);      // add dispose listeners
void removeEventListener (com::sun::star::lang::XEventListener aListener);  // remove them
```

com.sun.star.lang.XInitialization
This interface is used to allow other objects to use `createInstanceWithArguments()` or `createInstanceWithArgumentsAndContext()` with the component. It should be implemented and the arguments processed in `initialize()`:

```
// com::sun::star::lang::XInitialization

void initialize(sequence< any > aArguments) raises (com::sun::star::uno::Exception);
```

com.sun.star.lang.XMain
This interface is for use with the uno executable to instantiate the component independently from the OpenOffice.org service manager.

```
// com.sun.star.lang.XMain

long run (sequence< string > aArguments);
```

com.sun.star.uno.XAggregation
This interfaces makes the implementation cooperate in an aggregation. If implemented, other objects can aggregate to the implementation. Aggregated objects behave as if they were one. If another object aggregates the component, it holds the component and delegates calls to it, so that the component seems to be one with the aggregating object.

```
// com.sun.star.uno.XAggregation

void setDelegator(com.sun.star.uno.XInterface pDelegator);
any queryAggregation(type aType);
```

com.sun.star.lang.XUnoTunnel
This interface provides a pointer to the component to another component in the same process. This can be achieved with `XUnoTunnel`. `XUnoTunnel` should not be used by new components, because it is to be used for integration of existing implementations, if all else fails.

By now you should be able to decide which interfaces are interesting in your case. Sometimes the decision for or against an interface depends on the necessary effort as well. The following section discusses for each of the above interfaces how you can take advantage of pre-implemented helper classes in Java or C++, and what must happen in a possible implementation, no matter which language is used.

4.4.1 XInterface

All service implementations must implement `com.sun.star.uno.XInterface`. If a Java component is derived from a Java helper class that comes with the SDK, it supports `XInterface` automatically. Otherwise, it is sufficient to add `XInterface` or any other UNO interface to the `implements` list. The Java UNO runtime takes care of `XInterface`. In C++, there are helper classes to inherit that already implement `XInterface`. However, if `XInterface` is to be implemented manually, consider the code below.

The IDL specification for `com.sun.star.uno.XInterface` looks like this:

```
// module com::sun::star::uno
interface XInterface
{
    any queryInterface( [in] type aType );
    [oneway] void acquire();
    [oneway] void release();
};
```

Requirements for queryInterface()

When `queryInterface()` is called, the caller asks the implementation if it supports the interface specified by the type argument. The UNOIDL base type stores the name of a type and its `com.sun.star.uno.TypeClass`. The call must return an interface reference of the requested type if it is available or a void `any` if it is not. There are certain conditions a `queryInterface()` implementation must meet:

Constant Behaviour
If `queryInterface()` on a specific object has *once* returned a valid interface reference for a given type, it must *always* return a valid reference for any subsequent `queryInterface()` call for the same type on this object. A query for XInterface must always return the same reference.

If `queryInterface()` on a specific object has *once* returned a void `any` for a given type, it must *always* return a void `any` for the same type.

Symmetry
If `queryInterface()` for XBar on a reference xFoo returns a reference xBar, then `queryInterface()` on reference xBar for type XFoo must *return xFoo* or calls made on the returned reference must be *equivalent to calls to xFoo*.

Object Identity
In C++, two objects are the same if their `XInterface` are the same. The `queryInterface()` for `XInterface` will have to be called on both. In Java, check for the identity by calling the runtime function com.`sun.star.uni.UnoRuntime.areSame()`.

The reason for this specifications is that a UNO runtime environment may choose to cache `queryInterface()` calls. The rules are identical to the rules of the function `QueryInterface()` in MS COM.



If you want to implement `queryInterface()` in Java, for example, you want to export less interfaces than you implement, your class must implement the Java interface com.sun.star.uno.IQueryInterface.

Reference Counting

The methods `acquire()` and `release()` handle the lifetime of the UNO object. This is discussed in detail in chapter *3.3.7 Professional UNO - UNO Concepts - Lifetime of UNO Objects*. Acquire and release *must* be implemented in a thread-safe fashion. This is demonstrated in C++ in the section about C++ components below.

4.4.2 XTypeProvider

Every UNO object should implement the `com.sun.star.lang.XTypeProvider` interface.

Some applications need to know which interfaces an UNO object supports, for example, the OpenOffice.org Basic engine or debugging tools, such as the InstanceInspector. The `com.sun.star.lang.XTypeProvider` interface was introduced to avoid going through all known interfaces calling `queryInterface()` repetitively. The `XTypeProvider` interface is implemented by Java and C++ helper classes. If the `XTypeProvider` must be implemented manually, use the following methods:

```
// module com::sun::star::lang
interface XTypeProvider: com::sun::star::uno::XInterface
{
    sequence<type> getTypes();
    sequence<byte> getImplementationId();
};
```

The sections about Java and C++ components below show examples of `XTypeProvider` implementations.

Provided Types

The `com.sun.star.lang.XTypeProvider:getTypes()` method must return a list of types for all interfaces that `queryInterface()` provides. The OpenOffice.org Basic engine depends on this information to establish a list of method signatures that can be used with an object.

ImplementationID

For caching purposes, the `getImplementationId()` method has been introduced. The method must return a `byte` array containing an identifier for the implemented set of interfaces in this implementation class. It is important that one ID maps to one set of interfaces, but one set of interfaces can be known under multiple IDs. Every implementation class should generate a static ID.

4.4.3 XServiceInfo

Every service implementation should export the `com.sun.star.lang.XServiceInfo` interface. `XServiceInfo` must be implemented manually, because only the programmer knows what services the implementation supports. The sections about Java and C++ components below show examples for `XServiceInfo` implementations.

This is how the IDL specification for `XServiceInfo` looks like:

```
// module com::sun::star::lang
interface XServiceInfo: com::sun::star::uno::XInterface
{
    string getImplementationName();
    boolean supportsService( [in] string ServiceName );
    sequence<string> getSupportedServiceNames();
};
```

Implementation Name

The method `getImplementationName()` provides access to the *implementation name* of a service implementation. The implementation name uniquely identifies one implementation of service specifications in a UNO object. The name can be chosen freely by the implementation alone, because it does not appear in IDL. However, the implementation should adhere to the following naming conventions:

company prefix	dot	"comp"	dot	module name	dot	unique object name in module	*implemented service(s)*
com.sun.star	.	comp	.	forms	.	ODataBaseForm	*com.sun.star.forms.DataBaseForm*
org.openoffice	.	comp	.	test	.	OThumbs	*org.openoffice.test.ImageShrink* *org.openoffice.test.ThumbnailInsert* ...

If an object implements one single service, it can use the service name to derive an implementation name. Implementations of several services should use a name that describes the entire object.

If a `createInstance()` is called at the service manager using an *implementation name,* an instance of exactly that implementation is received. An implementation name is equivalent to a class name in Java. A Java component simply returns the fully qualified class name in `getImplementation-Name()`.



It is good practice to program against the specification and not against the implementation, otherwise, your application could break with future versions. OpenOffice.orgs API implementation is not supposed to be compatible, only the specification is.

Supported Service Names

The methods `getSupportedServiceNames()` and `supportsService()` deal with the availability of services in an implemented object. Note that the supported services are the services implemented in *one class* that supports these services, not the services of all implementations contained in the component file. If the illustration 34: A Component implementing three UNO objects, `XServiceInfo` is exported by the implemented objects in a component, not by the component. That means, srv3_4 must support `XServiceInfo` and return "Service3" and "Service4" as supported service names.

The service name identifies a service as it was specified in IDL. If an object is instantiated at the service manager using the service name, an object that complies to the service specification is returned.



The *single service factories* returned by components that are used to create instances of an implementation through their interfaces `com.sun.star.lang.XSingleComponentFactory` or `com.sun.star.lang.XSingleServiceFactory` must support XServiceInfo. The single factories support this interface to allow UNO to inspect the capabilities of a certain implementation before instantiating it. You can take advantage of this feature through the `com.sun.star.container.XContentEnumerationAccess` interface of a service manager.

4.4.4 XWeak

A component supporting `XWeak` offers other objects to hold a reference on itself without preventing it from being destroyed when it is no longer needed. Thus, cyclic references can be avoided easily. The chapter *3.3.7 Professional UNO - UNO Concepts - Lifetime of UNO Objects* discusses this in detail. In Java, derive from the Java helper class `com.sun.star.lib.uno.helper.WeakBase` to support `XWeak`. If a C++ component is derived from one of the `::cppu::Weak...ImplHelperNN` template classes as proposed in the section *4.6 Writing UNO Components - C++ Component*, a `XWeak` support is obtained, virtually for free. For the sake of completeness, this is the `XWeak` specification:

```
// module com::sun::star::uno::XWeak
```

```
interface XWeak: com::sun::star::uno::XInterface
{
    com::sun::star::uno::XAdapter queryAdapter();
};
```

4.4.5 XComponent

If the implementation holds a reference to another UNO object internally, there may be a problem of cyclic references that might prevent your component and the other object from being destroyed forever. If it is probable that the other object may hold a reference to your component, implement `com.sun.star.lang.XComponent` that contains a method `dispose()`. Chapter *3.3.7 Professional UNO - UNO Concepts - Lifetime of UNO Objects* discusses the intricacies of this issue.

Supporting `XComponent` in a C++ or Java component is simple, because there are helper classes to derive from that implement `XComponent`. The following code is an example if you must implement `XComponent` manually.

The interface `XComponent` specifies these operations:

```
// module com::sun::star::lang

interface XComponent: com::sun::star::uno::XInterface
{
    void dispose();
    void addEventListener( [in] XEventListener xListener );
    void removeEventListener( [in] XEventListener aListener );
};
```

`XComponent` uses the interface `com.sun.star.lang.XEventListener`:

```
// module com::sun::star::lang
interface XEventListener: com::sun::star::uno::XInterface
{
    void disposing( [in] com::sun::star::lang::EventObject Source );
};
```

Disposing of an XComponent

The idea behind `XComponent` is that the object is instantiated by a third object that makes the third object the *owner* of first object. The owner is allowed to call `dispose()`. When the owner calls `dispose()` at your object, it must do three things:

- Release all references it holds.
- Inform registered XEventListeners that it is being disposed of by calling their method `disposing()`.
- Behave as passive as possible afterwards. If the implementation is called after being disposed, throw a `com.sun.star.lang.DisposedException` if you cannot fulfill the method specification.

That way the owner of `XComponent` objects can dissolve a possible cyclic reference.

4.4.6 XInitialization

The interface `com.sun.star.lang.XInitialization` is usually implemented manually, because only the programmer knows how to initialize the object with arguments received from the service manager through `createInstanceWithArguments()` or `createInstanceWithArgumentsAndContext()`. In Java, `XInitialization` is used as well, but know that the Java factory helper provides a shortcut that uses arguments without implementing `XInitialization` directly. The Java factory

helper can pass arguments to the class constructor under certain conditions. Refer to the section *4.5.7 Writing UNO Components - Simple Component in Java - Create Instance With Arguments* for more information.

The specification for `XInitialization` looks like this:

```
// module com::sun::star::lang

interface XInitialization : com::sun::star::uno::XInterface
{
    void initialize(sequence< any > aArguments) raises (com::sun::star::uno::Exception);
};
```

Specify in the idl service specification which arguments and in which order are expected within the `any` sequence.

4.4.7 XMain

The implementation of `com.sun.star.lang.XMain` is used for special cases. Its `run()` operation is called by the *uno* executable. The section *4.10 Writing UNO Components - The UNO Executable* below discusses the use of `XMain` and the *uno* executable in detail.

```
// module com::sun::star::lang

interface XMain: com::sun::star::uno::XInterface
{
    long run( [in] sequence< string > aArguments );
};
```

4.4.8 XAggregation

A concept called *aggregation* is commonly used to plug multiple objects together to form one single object at runtime. The main interface in this context is `com.sun.star.uno.XAggregation`. After plugging the objects together, the reference count and the `queryInterface()` method is delegated from multiple *slave* objects to one *master* object.

It is a precondition that at the moment of aggregation, the slave object has a reference count of exactly one, which is the reference count of the master. Additionally, it does not work on proxy objects, because in Java, multiple proxy objects of the same interface of the same slave object might exist.

While aggregation allows more code reuse than implementation inheritance, the facts mentioned above, coupled with the implementation of independent objects makes programming prone to errors. Therefore the use of this concept is discourage and not explained here. For further information visit *http://udk.openoffice.org/common/man/concept/unointro.html#aggregation*.

4.4.9 XUnoTunnel

The `com.sun.star.lang.XUnoTunnel` interface allows access to the `this` pointer of an object. This interface is used to cast a UNO interface that is coming back to its implementation class through a UNO method. Using this interface is a result of an unsatisfactory interface design, because it indicates that some functionality only works when non-UNO functions are used. In general, these objects cannot be replaced by a different implementation, because they undermine the general UNO interface concept. This interface can be understood as admittance to an already existing code that cannot be split into UNO components easily. If designing new services, do not use this interface.

```
interface XUnoTunnel: com::sun::star::uno::XInterface
{
    hyper getSomething( [in] sequence< byte > aIdentifier );
};
```

The byte sequence contains an identifier that both the caller and implementer must know. The implementer returns the `this` pointer of the object if the byte sequence is equal to the byte sequence previously stored in a static variable. The byte sequence is usually generated *once per process* per implementation.



Note that the previously mentioned 'per process' is important because the `this` pointer of a class you know is useless, if the instance lives in a different process.

4.5 Simple Component in Java

This section shows how to write Java components. The examples in this chapter are in the samples folder that was provided with the programmer's manual.

A Java component is a library of Java classes (a jar) containing objects that implement arbitrary UNO services. For a service implementation in Java, implement the necessary UNO core interfaces and the interfaces needed for *your* purpose. These could be existing interfaces or interfaces defined by using UNOIDL.

Besides these service implementations, Java components need two methods to instantiate the services they implement in a UNO environment: one to get single factories for each service implementation in the jar, and another one to write registration information into a registry database. These methods are called *static component operations* in the following:

The method that provides single factories for the service implementations in a component is `__getServiceFactory()`:

```
public static XSingleServiceFactory __getServiceFactory(String implName,
                                    XMultiServiceFactory multiFactory,
                                    XRegistryKey regKey)
```

In theory, a client obtains a single factory from a component by calling `__getServiceFactory()` on the component implementation directly. This is rarely done because in most cases service manager is used to get an instance of the service implementation. The service manager uses `__getServiceFactory()` at the component to get a factory for the requested service from the component, then asks this factory to create an instance of the one object the factory supports.

To find a requested service implementation, the service manager searches its registry database for the location of the component jar that contains this implementation. For this purpose, the component must have been registered beforehand. UNO components are able to write the necessary information on their own through a function that performs the registration and which can be called by the registration tool *regcomp*. The function has this signature:

```
public static boolean __writeRegistryServiceInfo(XRegistryKey regKey)
```

These two methods work together to make the implementations in a component available to a service manager. The method `__writeRegistryServiceInfo()` tells the service manager where to find an implementation while `__getServiceFactory()` enables the service manager to instantiate a service implementation, once found.

The necessary steps to write a component are:

1. Define service implementation classes.
2. Implement UNO core interfaces.

3. Implement your own interfaces.
4. Provide static component operations to make your component available to a service manager.

4.5.1 Class Definition with Helper Classes

XInterface, XTypeProvider and XWeak

The OpenOffice.org Java UNO environment contains Java helper classes that implement the majority of the core interfaces that are implemented by UNO components. There are two helper classes:

- The helper com.sun.star.lib.uno.helper.WeakBase is the minimal base class and implements `XInterface`, `XTypeProvider` and `Xweak`.
- The helper com.sun.star.lib.uno.helper.ComponentBase that extends `WeakBase` and implements `XComponent`.

The `com.sun.star.lang.XServiceInfo` is the only interface that should be implemented, but it is not part of the helpers.

Use the naming conventions described in section *4.4.3 Writing UNO Components - Core Interfaces to Implement - XServiceInfo* for the service implementation. Following the rules, a service `org.openoffice.test.ImageShrink` should be implemented in `org.openoffice.`*`comp`*`.test.ImageShrink`.

A possible class definition that uses `ComponentBase` could look like this: (Components/Thumbs/org/openoffice/comp/test/ImageShrink.java)

```
package org.openoffice.comp.test;

public class ImageShrink extends com.sun.star.lib.uno.helper.ComponentBase
        implements com.sun.star.lang.XServiceInfo,
        org.openoffice.test.XImageShrink,
        com.sun.star.document.XFilter  {

    com.sun.star.uno.XComponentContext xComponentContext = null;

    /** Creates a new instance of ImageShrink */
    public ImageShrink(com.sun.star.uno.XComponentContext XComponentContext xContext) {
        this.xComponentContext = xContext;
    }
    ...
}
```

XServiceInfo

If the implementation only supports one service, use the following code to implement XServiceInfo: (Components/Thumbs/org/openoffice/comp/test/ImageShrink.java)

```
    ...

    //XServiceInfo implementation

    // hold the service name in a private static member variable of the class
    protected static final String __serviceName = "org.openoffice.test.ImageShrink";

    public String getImplementationName(  ) {
        return getClass().getName();
    }

    public boolean supportsService(String serviceName) {
        if ( serviceName.equals( __serviceName))
            return true;
```

```
        return false;
    }

    public String[] getSupportedServiceNames(  ) {
        String[] retValue= new String[0];
        retValue[0]= __serviceName;
        return retValue;
    }

    ...
```

An implementation of more than one service in one UNO object is more complex. It has to return all supported service names in `getSupportedServiceNames()`, furthermore it must check all supported service names in `supportsService()`. Note that several services packaged in one component file are not discussed here, but objects supporting more than one service. Refer to *34: A Component implementing three UNO objects* for the implementation of srv3_4.

4.5.2 Implementing your own Interfaces

The functionality of a component is accessible only by its interfaces. When writing a component, choose one of the available API interfaces or define an interface. IDL types are used as method arguments to other UNO objects. Java does not support unsigned data types, so their use is discouraged. In the chapter *4.2 Writing UNO Components - Using UNOIDL to Specify new Components*, the `org.openoffice.test.XImageShrink` interface specification was written and an interface class file was created. Its implementation is straightforward, you create a class that implements your interfaces: (Components/Thumbs/org/openoffice/comp/test/ImageShrink.java)

```
package org.openoffice.comp.test;

public class ImageShrink extends com.sun.star.lib.uno.helper.ComponentBase
        implements com.sun.star.lang.XServiceInfo,
                   org.openoffice.test.XImageShrink,
                   com.sun.star.document.XFilter {

    ...

    String destDir = "";
    String sourceDir = "";
    boolean cancel = false;
    com.sun.star.awt.Size dimension = new com.sun.star.awt.Size();

    // XFilter implementation

    public void cancel() {
        cancel = true;
    }

    public boolean filter(com.sun.star.beans.PropertyValue[] propertyValue) {
        // while cancel = false,
        // scale images found in sourceDir according to dimension and
        // write them to destDir, using the image file format given in
        // []propertyValue
         // (implementation omitted)
        cancel = false;
        return true;
    }

    // XIMageShrink implementation

    public String getDestinationDirectory() {
        return destDir;
    }

    public com.sun.star.awt.Size getDimension() {
        return dimension;
    }

    public String getSourceDirectory() {
        return sourceDir;
    }

    public void setDestinationDirectory(String str) {
        destDir = str;
    }
```

```
    public void setDimension(com.sun.star.awt.Size size) {
        dimension = size;
    }

    public void setSourceDirectory(String str) {
        sourceDir = str;
    }

    ...
}
```

For the component to run, the new interface class file must be accessible to the Java Virtual Machine. That is, it has to be in its CLASSPATH. All commonly used interfaces are contained in *ridl.jar* and unoil.jar that are always in the CLASSPATH because of the OpenOffice.org setup program.

The recommended method is to deliver the interface together with the component in the same jar file, or to have the interface in a separate jar or class file. In both cases, put the corresponding class with the interface into the CLASSPATH. This is achieved by editing the file *java(.ini|rc)* in *<office-path>\user\config* or through the options dialog. The *java(.ini|rc)* contains a `SystemClasspath` entry that you append the path pointing to the class or jar file. In the **Options** dialog, expand the OpenOffice.org node in the tree on the left-hand side and choose **Security**. One the right-hand side, there is a field **User Classpath** to add the jar or class file containing the interface.



It is also important that the binary type library of the new interfaces are provided together with the component, otherwise the component is not accessible from OpenOffice.org Basic. Basic uses the UNO core reflection service to get type information at runtime. The core reflection is based on the binary type library.

4.5.3 Providing a Single Factory Using Helper Method

The component must be able to create single factories for each service implementation it contains and return them in the static component operation `__getServiceFactory()`. The OpenOffice.org Java UNO environment provides a Java class `com.sun.star.comp.loader.FactoryHelper` that creates a default implementation of a single factory through its method `getServiceFactory()`. The following example could be written:
(Components/Thumbs/org/openoffice/comp/test/ImageShrink.java)

```
package org.openoffice.comp.test;

import com.sun.star.lang.XSingleServiceFactory;
import com.sun.star.lang.XMultiServiceFactory;
import com.sun.star.registry.XRegistryKey;
import com.sun.star.comp.loader.FactoryHelper;

public class ImageShrink ... {

  ...

    // static __getServiceFactory() implementation
    // static member __serviceName was introduced above for XServiceInfo implementation
    public static XSingleServiceFactory __getServiceFactory(String implName,
            XMultiServiceFactory multiFactory,
            com.sun.star.registry.XRegistryKey regKey) {

        com.sun.star.lang.XSingleServiceFactory xSingleServiceFactory = null;
        if (implName.equals( ImageShrink.class.getName()) )
            xSingleServiceFactory = FactoryHelper.getServiceFactory(ImageShrink.class,
                  ImageShrink.__serviceName, multiFactory, regKey);

        return xSingleServiceFactory;
    }

  ...

}
```

The `FactoryHelper` is contained in the *jurt* jar file. The `getServiceFactory()` method takes as a first argument a `Class` object. When `createInstance()` is called on the default factory, it creates

an instance of that `Class` using `newInstance()` on it and retrieves the implementation name through `getName()`. The second argument is the service name. The `multiFactory` and `regKey` arguments were received in `__getServiceFactory()` and are passed to the `FactoryHelper`.



In this case, the implementation name, which the default factory finds through `Class.getName()` is `org.openoffice.comp.test.ImageShrink` and the service name is `org.openoffice.test.ImageShrink`. The implementation name and the service name are used for the separate XServiceInfo implementation within the default factory. Not only do you support the XServiceInfo interface in your service implementation, but the single factory must implement this interface as well.

The default factory created by the `FactoryHelper` expects a public constructor in the implementation class of the service and calls it when it instantiates the service implementation. The constructor can be a default constructor, or it can take a `com.sun.star.uno.XComponentContext` or a `com.sun.star.lang.XMultiServiceFactory` as an argument. Refer to *4.5.7 Writing UNO Components - Simple Component in Java - Create Instance With Arguments* for other arguments that are possible.

Java components are housed in jar files. When a component has been registered, the registry contains the name of the jar file, so that the service manager can find it. However, because a jar file can contain several class files, the service manager must be told which one contains the `__getServiceFactory()` method. That information has to be put into the jar's Manifest file, for example:

```
RegistrationClassName: org.openoffice.comp.test.ImageShrink
```

4.5.4 Write Registration Info Using Helper Method

UNO components have to be registered with the registry database of a service manager. In an office installation, this is the file *applicat.rdb* for all predefined services. A service manager can use this database to find the implementations for a service. For instance, if an instance of your component is created using the following call.

```
Object imageShrink = xRemoteServiceManager.createInstance("org.openoffice.test.ImageShrink");
```

Using the given service or implementation name, the service manager looks up the location of the corresponding jar file in the registry and instantiates the component.



If you want to use the service manager of the Java UNO runtime, com.sun.star.comp.servicemanager.ServiceManager (jurt.jar), to instantiate your service implementation, then you would have to create the service manager and add the factory for “org.openoffice.test.ImageShrink” programmatically, because the Java service manager does not use the registry.

Alternatively, you can use com.sun.star.comp.helper.RegistryServiceFactory from juh.jar which is registry-based. Its drawback is that it delegates to a C++ implementation of the service manager through the java-bridge.

During the registration, a component writes the necessary information into the registry. The process to write the information is triggered externally when a client calls the `__writeRegistryServiceInfo()` method at the component.

```
public static boolean __writeRegistryServiceInfo(XRegistryKey regKey)
```

The caller passes an `com.sun.star.registry.XRegistryKey` interface that is used by the method to write the registry entries. Again, the FactoryHelper class offers a way to implement the method: (Components/Thumbs/org/openoffice/comp/test/ImageShrink.java)

```
    ...

    // static __writeRegistryServiceInfo implementation
    public static boolean __writeRegistryServiceInfo(XRegistryKey regKey) {
```

```
        return FactoryHelper.writeRegistryServiceInfo( ImageShrink.class.getName(),
            __serviceName, regKey);
    }
```

The writeRegistryServiceInfo method takes three arguments:

- implementation name
- service name
- XRegistryKey

Use tools, such as *regcomp* or the Java application *com.sun.star.tools.uno.RegComp* to register a component. These tools take the path to the jar file containing the component as an argument. Since the jar can contain several classes, the class that implements the `__writeRegistryServiceInfo()` method must be pointed out by means of the manifest. Again, the `RegistrationClassName` entry determines the correct class. For example:

```
RegistrationClassName: org.openoffice.comp.test.ImageShrink
```

The above entry is also necessary to locate the class that provides `__getServiceFactory()`, therefore the functions `__writeRegistryServiceInfo()` and `__getServiceFactory()` have to be in the same class.

4.5.5 Implementing without Helpers

XInterface

As soon as the component implements any UNO interface, `com.sun.star.uno.XInterface` is included automatically. The Java interface definition generated by *javamaker* for `com.sun.star.uno.XInterface` contains a `TypeInfo` member used by Java UNO internally to store certain IDL type information (Refer to *3.4.1 Professional UNO - UNO Language Bindings - Java Language Binding*):

```
// source file com/sun/star/uno/XInterface.java generated by javamaker

package com.sun.star.uno;

public interface XInterface
{
    // static Member
    public static final com.sun.star.lib.uno.typeinfo.TypeInfo UNOTYPEINFO[] = null;
}
```

Note that `XInterface` does not have any methods, in contrast to its IDL description. That means, if `implements com.sun.star.uno.XInterface` is added to a class definition, there is nothing to implement.

The method `queryInterface()` is unnecessary in a service implementation, because the Java UNO runtime environment obtains interface references without being helped by the components. Within Java, the method `UnoRuntime.queryInterface()` is used to obtain interfaces instead of calling `com.sun.star.uno.XInterface:queryInterface()`, and the Java UNO language binding hands out interfaces for services to other processes on its own as well.

The methods `acquire()` and `release()` are used for reference counting and control the lifetime of an object, because the Java garbage collector does this, there is no reference counting in Java components.

XTypeProvider

Helper classes with default `com.sun.star.lang.XTypeProvider` implementations are still under development for Java. Meanwhile, every Java UNO object implementation can implement the `XTypeProvider` interface as shown in the following code. In your implementation, adjust `getTypes()`: (Components/Thumbs/org/openoffice/comp/test/ImageShrink.java)

```
...

// XTypeProvider implementation

// maintain a static implementation id for all instances of ImageShrink
// initialized by the first call to getImplementationId()
protected static byte[] _implementationId;

public com.sun.star.uno.Type[] getTypes() {
    com.sun.star.uno.Type[] retValue = new com.sun.star.uno.Type[4];

    // instantiate Type instances for each interface you support and add them to Type[] array

    // this object implements XServiceInfo, XTypeProvider and XImageShrink
    retValue[0]= new com.sun.star.uno.Type( com.sun.star.lang.XServiceInfo.class);
    retValue[1]= new com.sun.star.uno.Type( com.sun.star.lang.XTypeProvider.class);
    retValue[3]= new com.sun.star.uno.Type( com.sun.star.document.XFilter);
    retValue[2]= new com.sun.star.uno.Type( org.openoffice.test.XImageShrink.class);
    // inherited interfaces, like XInterface, are recognized implicitely

    return retValue;
}

synchronized public byte[] getImplementationId() {
    if (_implementationId == null) {
        _implementationId= new byte[16];
        int hash = hashCode();  // hashCode of this object
        _implementationId[0] = (byte)(hash & 0xff);
        _implementationId[1] = (byte)((hash >>> 8) & 0xff);
        _implementationId[2] = (byte)((hash >>> 16) & 0xff);
        _implementationId[3] = (byte)((hash >>>24) & 0xff);
    }
    return _implementationId;
}

...
```

The suggested implementation of the `getImplementationId()` method is not optimal, it uses the `hashCode()` of the first instance that initializes the static field. The future UNO helper class will improve this.

XComponent

`XComponent` is an optional interface that is useful when other objects hold references to the component. The notification mechanism of `XComponent` enables listener objects to learn when the component stops to provide its services, so that the objects drop their references to the component. This enables the component to delete itself when its reference count drops to zero. From section *4.4 Writing UNO Components - Core Interfaces to Implement*, there must be three things done when `dispose()` is called at an XComponent:

- Inform registered `XEventListener`s that the object is being disposed of by calling their method `disposing()`.
- Release all references the object holds, including all `XEvenListener` objects.
- On further calls to the component, throw an `com.sun.star.lang.DisposedException` in case the required task can not be fulfilled anymore, because the component was disposed.

In Java, the object cannot be deleted, but the garbage collector will do this. It is sufficient to release all references that are currently being held to break the cyclic reference, and to call `disposing()` on all `com.sun.star.lang.XEventListeners`.

The registration and removal of listener interfaces is a standard procedure in Java. Some IDEs even create the necessary methods automatically. The following example could be written: (Components/Thumbs/org/openoffice/comp/test/ImageShrink.java)

```
...

//XComponent implementation

// hold a Vector of eventListeners in the class
private transient Vector eventListeners;

void dispose {
    fireDisposing(new com.sun.star.lang.EventObject(this))
    releaseReferences();
}

public synchronized void addEventListener(XEventListener listener) {

    if ( eventListeners == 0 )
        eventListeners = new Vector(2);
    if ( !eventListeners.contains(listener) )
        eventListeners.addElement(listener);
}

public synchronized void removeEventListener(XEventListener listener) {

    if ( eventListeners != 0 )
        eventListeners.removeElement(listener);
}


protected void fireDisposing(com.sun.star.lang.EventObject e) {
    if (eventListeners != null) {
        Vector listeners = eventListeners ;
        int count = listeners.size();
        for (int i = 0; i < count; i++) {
            ((XEventListener) listeners.elementAt(i)).disposing(e);
        }
    }
}

protected void releaseReferences() {
    xComponentContext = null;
    // ...
}

...
```

4.5.6 Storing the Service Manager for Further Use

A component usually runs in the office process. There is no need to create an interprocess channel explicitly. A component does not have to create a service manager, because it is provided to the single factory of an implementation by the service manager during a call to `createInstance()` or `createInstanceWithContext()`. The single factory receives an `XComponentContext` or an `XMultiServiceFactory`, and passes it to the corresponding constructor of the service implementation. From the component context, the implementation gets the service manager using `getServiceManager()` at the `com.sun.star.uno.XComponentContext` interface .

4.5.7 Create Instance with Arguments

A factory can create an instance of components and pass additional arguments. To do that, a client calls the `createInstanceWithArguments()` function of the `com.sun.star.lang.XSingleServiceFactory` interface or the `createInstanceWithArgumentsAndContext()` of the `com.sun.star.lang.XSingleComponentFactory` interface.

```
//javamaker generated interface
//XSingleServiceFactory interface
public java.lang.Object createInstanceWithArguments(java.lang.Object[] aArguments)
                            throws com.sun.star.uno.Exception;
```

```
//XSingleComponentFactory
public java.lang.Object createInstanceWithArgumentsAndContext(java.lang.Object[] Arguments,
                                com.sun.star.uno.XComponentContext Context)
                            throws com.sun.star.uno.Exception;
```

Both functions take an array of values as an argument. A component implements the `com.sun.star.lang.XInitialization` interface to receive the values. A factory passes the array on to the single method `initialize()` supported by XInitialization.

```
public void initialize(java.lang.Object[] aArguments) throws com.sun.star.uno.Exception;
```

Alternatively, a component may also receive these arguments in its constructor. If a factory is written, determine exactly which arguments are provided by the factory when it instantiates the component. When using the `FactoryHelper`, implement the constructors with the following arguments:

First Argument	Second Argument	Third Argument
com.sun.star.uno.XComponentContext	com.sun.star.registry.XRegistryKey	java.lang.Object[]
com.sun.star.uno.XComponentContext	com.sun.star.registry.XRegistryKey	
com.sun.star.uno.XComponentContext	java.lang.Object[]	
com.sun.star.uno.XComponentContext		
java.lang.Object[]		

The `FactoryHelper` automatically passes the array of arguments it received from the `createInstanceWithArguments[AndContext]()` call to the appropriate constructor. Therefore, it is not always necessary to implement `XInitialization` to use arguments.

4.5.8 Possible Structures for Java Components

The implementation of a component depends on the needs of the implementer. The following examples show some possible ways to assemble a component. There can be one implemented object or several implemented objects per component file.

One Implementation per Component File

There are additional options if implementing one service per component file:

- Use a flat structure with the static component operations added to the service implementation class directly.
- Reserve the class with the implementation name for the static component operation and use an inner class to implement the service.

Implementation Class with Component Operations

An implementation class contains the static component operations. The following sample implements an interface `com.sun.star.test.XSomething` in an implementation class `JavaComp.TestComponent`:

```
// UNOIDL: interface example specification
module com {  module sun {  module star {  module test {

interface XSomething: com::sun::star::uno::XInterface
{
    string methodOne([in]string val);
};
}; }; }; };
```

A component that implements only one service supporting `XSomething` can be assembled in one class as follows:

```
package JavaComp;

...

public class TestComponent implements XSomething, XTypeProvider, XServiceInfo {

    public static final String __serviceName="com.sun.star.test.JavaTestComponent";

    public static XSingleServiceFactory __getServiceFactory(String implName,
                                XMultiServiceFactory multiFactory, XRegistryKey regKey) {
    XSingleServiceFactory xSingleServiceFactory = null;

    if (implName.equals( TestComponent.class.getName()) )
        xSingleServiceFactory = FactoryHelper.getServiceFactory( TestComponent.class,
                                TestComponent.__serviceName, multiFactory, regKey);
        return xSingleServiceFactory;
    }

    public static boolean __writeRegistryServiceInfo(XRegistryKey regKey){
        return FactoryHelper.writeRegistryServiceInfo( TestComponent.class.getName(),
                    TestComponent.__serviceName, regKey);
    }

    // XSomething
    string methodOne(String val) {
        return val;
    }
    //XTypeProvider
    public com.sun.star.uno.Type[] getTypes(  ) {
        ...
    }
    // XTypeProvider
    public byte[] getImplementationId(  ) {
        ...
    }
    //XServiceInfo
    public String getImplementationName(  ) {
        ...
    }
    // XServiceInfo
    public boolean supportsService( /*IN*/String serviceName ) {
        ...
    }
    //XServiceInfo
    public String[] getSupportedServiceNames(  ) {
        ...
    }
}
```

The class implements the `XSomething` interface. The IDL description and documentation provides information about its functionality. The class also contains the functions for factory creation and registration, therefore the manifest entry must read as follows:

```
RegistrationClassName: JavaComp.TestComponent
```

Implementation Class with Component Operations and Inner Implementation Class

To implement the component as inner class of the one that provides the service factory through `__getServiceFactory()`, it must be a *static* inner class, otherwise the factory provided by the `FactoryHelper` cannot create the component. An example for an inner implementation class is located in the sample com.sun.star.comp.demo.DemoComponent.java provided with the SDK. The implementation of `__getServiceFactory()` and `__writeRegistryServiceInfo()` is omitted here, because they act the same as in the implementation class with component operations above.

```
package com.sun.star.comp.demo;

public class DemoComponent {

    ...
    // static inner class implements service com.sun.star.demo.DemoComponent
    static public class _Implementation implements XTypeProvider,
                            XServiceInfo, XInitialization, XWindowListener,
                            XActionListener, XTopWindowListener {

        static private final String __serviceName = "com.sun.star.demo.DemoComponent";
```

```
        private XMultiServiceFactory _xMultiServiceFactory;

        // Constructor
        public _Implementation(XMultiServiceFactory xMultiServiceFactory) {
        }
    }

    // static method to get a single factory creating the given service from the factory helper
    public static XSingleServiceFactory __getServiceFactory(String implName,
                                            XMultiServiceFactory multiFactory,
                                            XRegistryKey regKey) {
         ...
    }

    // static method to write the service information into the given registry key
    public static boolean __writeRegistryServiceInfo(XRegistryKey regKey) {
        ...
    }

}
```

The manifest entry for this implementation structure again has to point to the class with the static component operations:

```
RegistrationClassName: com.sun.star.comp.demo.DemoComponent
```

Multiple Implementations per Component File

To assemble several service implementations in one component file, implement each service in its own class and add a separate class containing the static component operations. The following code sample features two services: `TestComponentA` and `TestComponentB` implementing the interfaces `XSomethingA` and `XSomethingB` with a separate static class `TestServiceProvider` containing the component operations.

The following are the UNOIDL specifications for `XSomethingA` and `XSomethingB`:

```
module com {  module sun {  module star {  module test {
interface XSomethingA: com::sun::star::uno::XInterface
{
    string methodOne([in]string value);
};
}; }; }; };

module com {  module sun {  module star {  module test {
interface XSomethingB: com::sun::star::uno::XInterface
{
    string methodTwo([in]string value);
};
}; }; }; };
```

`TestComponentA` implements `XSomethingA`:
(Components/JavaComponent/TestComponentA.java):

```
package JavaComp;

public class TestComponentA implements XTypeProvider, XServiceInfo, XSomethingA {
    static final String __serviceName= "JavaTestComponentA";

    static byte[] _implementationId;

    public TestComponentA() {
    }

    // XSomethingA
    public String methodOne(String val) {
        return val;
    }

    //XTypeProvider
    public com.sun.star.uno.Type[] getTypes(  ) {
        Type[] retValue= new Type[3];
        retValue[0]= new Type( XServiceInfo.class);
        retValue[1]= new Type( XTypeProvider.class);
        retValue[2]= new Type( XSomethingA.class);
        return retValue;
    }

    //XTypeProvider
```

```
    synchronized public byte[] getImplementationId(  ) {
        if (_implementationId == null) {
            _implementationId= new byte[16];
            int hash = hashCode();
            _implementationId[0] = (byte)(hash & 0xff);
            _implementationId[1] = (byte)((hash >>> 8) & 0xff);
            _implementationId[2] = (byte)((hash >>> 16) & 0xff);
            _implementationId[3] = (byte)((hash >>>24) & 0xff);
        }
        return _implementationId;
    }

    //XServiceInfo
    public String getImplementationName(  ) {
        return getClass().getName();
    }
    // XServiceInfo
    public boolean supportsService( /*IN*/String serviceName ) {
        if ( serviceName.equals( __serviceName))
            return true;
        return false;
    }

    //XServiceInfo
    public String[] getSupportedServiceNames(  ) {
        String[] retValue= new String[0];
        retValue[0]= __serviceName;
        return retValue;
    }
}
```

`TestComponentB` implements `XSomethingB`. Note that it receives the component context and initialization arguments in its constructor. (Components/JavaComponent/TestComponentB.java)

```
package JavaComp;

public class TestComponentB implements XTypeProvider, XServiceInfo, XSomethingB {
    static final String __serviceName= "JavaTestComponentB";

    static byte[] _implementationId;
    private XComponentContext context;
    private Object[] args;

    public TestComponentB(XComponentContext context, Object[] args) {
        this.context= context;
        this.args= args;
    }

    // XSomethingB
    public String methodTwo(String val) {
        if (args.length > 0 && args[0] instanceof String )
            return (String) args[0];
        return val;
    }

    //XTypeProvider
    public com.sun.star.uno.Type[] getTypes(  ) {
        Type[] retValue= new Type[3];
        retValue[0]= new Type( XServiceInfo.class);
        retValue[1]= new Type( XTypeProvider.class);
        retValue[2]= new Type( XSomethingB.class);
        return retValue;
    }

    //XTypeProvider
    synchronized public byte[] getImplementationId(  ) {
        if (_implementationId == null) {
            _implementationId= new byte[16];
            int hash = hashCode();
            _implementationId[0] = (byte)(hash & 0xff);
            _implementationId[1] = (byte)((hash >>> 8) & 0xff);
            _implementationId[2] = (byte)((hash >>> 16) & 0xff);
            _implementationId[3] = (byte)((hash >>>24) & 0xff);
        }
        return _implementationId;
    }

    //XServiceInfo
    public String getImplementationName(  ) {
        return getClass().getName();
    }

    // XServiceInfo
    public boolean supportsService( /*IN*/String serviceName ) {
```

```
        if ( serviceName.equals( __serviceName))
            return true;
        return false;
    }

    //XServiceInfo
    public String[] getSupportedServiceNames(  ) {
        String[] retValue= new String[0];
        retValue[0]= __serviceName;
        return retValue;
    }
}
```

`TestServiceProvider` implements `__getServiceFactory()` and `__writeRegistryServiceInfo()`: (Components/JavaComponent/TestServiceProvider.java)

```
package JavaComp;
...
public class TestServiceProvider
{
    public static XSingleServiceFactory __getServiceFactory(String implName,
                                        XMultiServiceFactory multiFactory,
                                        XRegistryKey regKey) {
        XSingleServiceFactory xSingleServiceFactory = null;

        if (implName.equals( TestComponentA.class.getName()) )
            xSingleServiceFactory = FactoryHelper.getServiceFactory( TestComponentA.class,
                                    TestComponentA.__serviceName, multiFactory, regKey);
        else if (implName.equals(TestComponentB.class.getName()))
            xSingleServiceFactory= FactoryHelper.getServiceFactory( TestComponentB.class,
                                    TestComponentB.__serviceName, multiFactory, regKey);
        return xSingleServiceFactory;
    }

    public static boolean __writeRegistryServiceInfo(XRegistryKey regKey){
        boolean bregA= FactoryHelper.writeRegistryServiceInfo( TestComponentA.class.getName(),
                        TestComponentA.__serviceName, regKey);
        boolean bregB= FactoryHelper.writeRegistryServiceInfo( TestComponentB.class.getName(),
                        TestComponentB.__serviceName, regKey);
        return bregA && bregB;
    }
}
```

The corresponding manifest entry must point to the static class with the component operations, in this case `JavaComp.TestServiceProvider`:

```
RegistrationClassName: JavaComp.TestServiceProvider
```

4.5.9 Running and Debugging Java Components

Registration

The service manager with a registry database containing information about the location of the component file and the types used must be provided to test the component with the office. There are several ways to accomplish the registration. The following is a possibility.

A *.rdb* file is created with all the necessary information and the service manager is told to use it in addition to the standard *applicat.rdb*. The advantage of proceeding this way is that the *applicat.rdb* does not become cluttered with the production office installation with test registrations and abandoned type information. Follow the steps below:

Note, it errors are encountered, refer to the troubleshooting section at the end of this chapter.

Register Component File

This step creates a registry file that contains the location of the component file and all the necessary type information. To register, place a few files to the proper locations:

- Copy the *regcomp* tool from the SDK distribution to *<OfficePath>/program*.

- Copy the component jar to *<OfficePath>/program/classes*.
- Copy the *.rdb* file containing the new types created to *<OfficePath>/program*. If new types were not defined, dismiss this step. In this case, *regcomp* automatically creates a new rdb file with registration information.

On the command prompt, change to *<OfficePath>/program*, then run *regcomp* with the following options. Line breaks were applied to improve readability, but the command must be entered in a single line:

```
$ regcomp -register -r <your_registry>.rdb
                    -br applicat.rdb
                    -l com.sun.star.loader.Java2
                    -c file:///<OfficePath>/program/classes/<your_component>.jar
```

For the `org.openoffice.test.ImageShrink` service whose type description was merged into *thumbs.rdb* , which is implemented in *thumbs.jar*, the corresponding command would be:

```
$ regcomp -register -r thumbs.rdb
                    -br applicat.rdb
                    -l com.sun.star.loader.Java2
                    -c file:///i:/StarOffice6.0/program/classes/thumbs.jar
```

Instead of *regcomp,* there is also a Java tool to register components, however, it can only write to the same registry it reads from. It cannot be used to create a separate registry database. For details, see the section *4.9 Writing UNO Components - Deployment Options for Components*.

Make Registration available to OpenOffice.org

OpenOffice.org must be told to use the registry. Close all OpenOffice.org parts, including the Quickstarter that runs in the Windows task bar. Edit the file *uno(.ini|rc)* in *<OfficePath>/program* as follows:

```
[Bootstrap]
UNO_TYPES=$SYSBINDIR/applicat.rdb $SYSBINDIR/<your_registry>.rdb
UNO_SERVICES=$SYSBINDIR/applicat.rdb $SYSBINDIR/<your_registry>.rdb
```

For details about the syntax of *uno(.ini|rc)* and alternative registration procedures, refer to the section *4.9 Writing UNO Components - Deployment Options for Components*. If OpenOffice.org is restarted, the component should be available.

Test the Registration

A short OpenOffice.org Basic program indicates if the program runs went smoothly, by selecting **Tools** – **Macro** and entering a new macro name on the left, such as `TestImageShrink` and click **New** to create a new procedure. In the procedure, enter the appropriate code of the component. The test routine for `ImageShrink` would be:

```
Sub TestImageShrink
    oTestComp = createUnoService("org.openoffice.test.ImageShrink")
    MsgBox oTestComp.dbg_methods
    MsgBox oTestComp.dbg_properties
    MsgBox oTestComp.dbg_supportedInterfaces
end sub
```

The result should be three dialogs showing the methods, properties and interfaces supported by the implementation. Note that the interface attributes do not appear as get/set methods, but as properties in Basic. If the dialogs do not show what is expected, refer to the section *4.5.9 Writing UNO Components - Simple Component in Java - Testing and Debugging Java Components - Troubleshooting*.

Debugging

To increase turnaround cycles and source level debugging, configure the IDE to use GNU make-files for code generation and prepare OpenOffice.org for Java debugging. If NetBeans are used, the following steps are necessary:

Support for GNU make

A NetBeans extension, available on *makefile.netbeans.org,* that adds basic support for GNU makefiles. When it is enabled, edit the makefile in the IDE and use the makefile to build. To install and enable this module, select **Tools – Setup Wizard** and click **Next** to go to the Module installation page. Find the module **Makefiles** and change the corresponding entry to True in the **Enabled** column. Finish using the setup wizard. If the module is not available in the installation, use **Tools – Update Center** to get the module from *www.netbeans.org.* A new entry, **Makefile Support,** appears in the online help when **Help – Contents** is selected. Makefile Support provides further configuration options. The settings **Run a Makefile** and **Test a Makefile** can be found in **Tools** – **Options** – **Uncategorized** – **Compiler Types** and – **Execution Types**.

Put the makefile into the project source folder that was mounted when the project was created. To build the project using the makefile, highlight the makefile in the **Explorer** and press **F11**.

Documentation for GNU *make* command- line options and syntax are available at www.gnu.org. The sample *Thumbs* in the samples folder along with this manual contains a makefile that with a few adjustments is useful for Java components.

Component Debugging

If NetBeans or Forte for Java is used, the Java Virtual Machine (JVM) that is launched by OpenOffice.org can be attached. Configure the JVM used by OpenOffice.org to listen for debugger connections. First close any open OpenOffice.org windows including the Quickstarter, then edit the section [JAVA] of the file *java(.ini|rc)* in *<OfficePath>/user/config* by adding:

```
-Xdebug
-Xrunjdwp:transport=dt_socket,server=y,address=8000,suspend=n
```

The last line causes the JVM to listen for a debugger on port 8000. The JVM starts listening as soon as it runs and does not wait until a debugger connects to the JVM. Launch the office and instantiate the Java component, so that the office invokes the JVM in listening mode.

Once a Java component is instantiated, the JVM keeps listening even if the component goes out of scope. Open the appropriate source file in the NetBeans editor and set breakpoints as needed. Choose **Debug - Attach**, select **Java Platform Debugger Architecture (JPDA)** as debugger type and **SocketAttach (Attaches by socket to other VMs)** as the connector. The **Host** should be localhost and the **Port** must be 8000. Click **OK** to connect the Java Debugger to the JVM the office has started previously step.

Once the debugger connects to the running JVM, NetBeans switches to debug mode, the output windows shows a message that a connection on port 8000 is established and threads are visible, as if the debugging was local. If necessary, start your component once again. As soon as the component reaches a breakpoint in the source code, the source editor window opens with the breakpoint highlighted by a green arrow.

The Java Environment in OpenOffice.org

When UNO components written in Java are to be used within the office, it has to be configured appropriately. During OpenOffice.org installation, the Java setup is run. It gives the user the opportunity to choose a Java installation. The setup only offers Java versions which are certain to work with the office. The user can also choose to have an appropriate Java Runtime Environment installed. When the office has been installed, a user can still change the used Java installation by running the *jvmsetup* program that is located in the program directory of the installation directory. For example:

d:\program files\<office-installation-dir>\program\jvmsetup.exe

When the office starts Java, it uses configuration data that are kept in the *java(.ini|rc)* file, as well as in dedicated configuration files. The *java(.ini|rc)* is located in the *<officepath>\user\config directory*. A client can use that file to pass additional properties to the Java Virtual Machine, which are then available as system properties. For example, to pass the property `MyAge`, invoke Java like this:

```
java -DMyAge=30 RunClass
```

If you want to have that system property accessible by your Java component you can put that property into *java(.ini|rc)* within the `[Java]` section. For example:

```
[Java]
Home=d:\development\jdk1.3.1

VMType=jdk
Version=1.3.1
RuntimeLib=d:\development\jdk1.3.1\jre\bin\hotspot\jvm.dll
SystemClasspath=d:\development\jdk1.3.1\jre\lib\rt.jar; ...
Java=1
JavaScript=1
Applets=1
MyAge=27
```

To debug a Java component, it is necessary to start the JVM with additional parameters. The parameters can be put in the *java.ini* the same way as they would appear on the command-line. For example , add those lines to the `[Java]` section:

```
-Xdebug
-Xrunjdwp:transport=dt_socket,server=y,address=8000
```

More about debugging can be found in the JDK documentation and in the OpenOffice.org Software Development Kit.

Java components are also affected by the following configuration settings. They can be changed in the **Tools - Options** dialog. In the dialog, expand the OpenOffice.org node on the left-hand side and choose **Security**. This brings up a new pane on the right-hand side that allows Java specific settings:

Java Setting	Description
Enable	If checked, Java is used with the office. This affects Java components, as well as applets.
Security checks	If checked, the security manager restricts resource access of applets.
Net access	Determines where an applet can connect.
ClassPath	Additional jar files and directories where the JVM should search for classes. Also known as user classpath.
Applets	If checked, applets are executed.

Troubleshooting

If the component encounters problems, review the following checklist to check if the component is configured correctly.

Check Registry Keys

To check if the registry database is correctly set up, run *regview* against the three keys that make up a registration in the /UCR, /SERVICES and /IMPLEMENTATIONS branch of a registry database. The following examples show how to read the appropriate keys and how a proper configuration should look. In our example, service `ImageShrink`, and the key /UCR/org/openoffice/test/XImageShrink contain the type information specified in UNOIDL:

```
# dump XImageShrink type information
```

```
$ regview thumbs.rdb /UCR/org/openoffice/test/XImageShrink

Registry "file:///X:/office60eng/program/thumbs.rdb":

/UCR/org/openoffice/test/XImageShrink
 Value: Type = RG_VALUETYPE_BINARY
        Size = 364
        Data = minor version: 0
               major version: 1
               type: 'interface'
               uik: { 0x00000000-0x0000-0x0000-0x00000000-0x00000000 }
               name: 'org/openoffice/test/XImageShrink'
               super name: 'com/sun/star/uno/XInterface'
               Doku: ""
               IDL source file: "X:\SO\sdk\examples\java\Thumbs\org\openoffice\test\XImageShrink.idl"
               number of fields: 3
               field #0:
                 name='SourceDirectory'
                 type='string'
                 access=READWRITE
                  Doku: ""
                 IDL source file: ""
               field #1:
                 name='DestinationDirectory'
                 type='string'
                 access=READWRITE
                  Doku: ""
                 IDL source file: ""
               field #2:
                 name='Dimension'
                 type='com/sun/star/awt/Size'
                 access=READWRITE
                  Doku: ""
                 IDL source file: ""
               number of methods: 0
               number of references: 0
```

The /SERVICES/org.openoffice.test.ImageShrink key must point to the implementation name `org.openoffice.`*`comp`*`.test.ImageShrink` that was chosen for this service:

```
# dump service name registration

$ regview thumbs.rdb /SERVICES/org.openoffice.test.ImageShrink

Registry "file:///X:/office60eng/program/thumbs.rdb":

/SERVICES/org.openoffice.test.ImageShrink
 Value: Type = RG_VALUETYPE_STRINGLIST
        Size = 45
        Len  = 1
        Data = 0 = "org.openoffice.comp.test.ImageShrink"
```

Finally, the /IMPLEMENTATIONS/org.openoffice.comp.test.ImageShrink key must contain the loader and the location of the component jar:

```
# dump implementation name registration

$ regview thumbs.rdb /IMPLEMENTATIONS/org.openoffice.comp.test.ImageShrink

Registry "file:///X:/office60eng/program/thumbs.rdb":

/IMPLEMENTATIONS/org.openoffice.comp.test.ImageShrink
 / UNO
   / ACTIVATOR
     Value: Type = RG_VALUETYPE_STRING
            Size = 26
            Data = "com.sun.star.loader.Java2"
   / SERVICES
     / org.openoffice.test.ImageShrink
   / LOCATION
     Value: Type = RG_VALUETYPE_STRING
            Size = 50
            Data = "file:///X:/office60eng/program/classes/thumbs.jar"
```

If the UCR key is missing, the problem is with *regmerge*. The most probable cause are missing *.urd* files. Be careful when writing the makefile. If *.urd* files are missing when regmerge is launched by the makefile, *regmerge* continues and creates a barebone *.rdb* file, sometimes without any type info.

If *regview* can not find the /SERVICES and /IMPLEMENTATIONS keys or they have the wrong content, the problem occurred when *regcomp* was run. This can be caused by wrong path names in the *regcomp* arguments.

Also, a wrong `SystemClasspath` setup in *java(.ini|rc)* could be the cause of *regcomp* error messages about missing classes. Check what the `SystemClasspath` entry in *java(.ini|rc)* specifies for the Java UNO runtime jars.

Ensure that *regcomp* is being run from the current directory when registering Java components. In addition, ensure *<OfficePath>/program* is the current folder when *regcomp* is run. Verify that *regcomp* is in the current folder.

Check the Java VM settings

Whenever the VM service is instantiated by OpenOffice.org, it uses the Java configuration settings in OpenOffice.org. This happens during the registration of Java components, therefore make sure that Java is enabled. Choose **Tools-Options** in OpenOffice.org, so that the dialog appears. Expand the OpenOffice.org node and select **Security**. Select the **Enable** checkbox in the Java section and click **OK**.

Check the Manifest

Make sure the manifest file contains the correct entry for the registration class name. The file must contain the following line:

```
RegistrationClassName: <full name of package and class>
```

The registration class name must be the one that implements the `__writeRegistryServiceInfo()` and `__getServiceFactory()` methods. The `RegistrationClassName` to be entered in the manifest for our example is `org.openoffice.comp.test.ImageShrink`.

Adjust CLASSPATH for Additional Classes

OpenOffice.org maintains its own system classpath and a user classpath when it starts the Java VM for Java components. The jar file that contains the service implementation is not required in the system or user classpath. If *other* jar files or classes are depended on and they are not part of the Java UNO runtime jars, they must be in the classpath. To correct the problem, edit the *java(.ini|rc)* file in *<OfficePath>/user/config* and add the jars or directories to the `SystemClasspath` entry or use **Tools – Options – OpenOffice.org - Security** to add them to the user classpath.

Disable Debug Options

If the debug options (-Xdebug, -Xrunjdwp) are in the *java(.ini|rc)* file, disable them by putting semicolons at the beginning of the respective lines. The *regcomp* may hang, because the JVM is waiting for a debugger to be attached.

4.6 C++ Component

In this section, a sample component containing two service implementations with helpers and without helpers implemented are presented. The complete source code and the gnu makefile are in *samples/simple_cpp_component*.

The first step for the C++ component is to define a language-independent interface, so that the UNO object can communicate with others. The IDL specification for the component defines one interface `my_module.XSomething` and two services implementing this interface. In addition, the second service called `my_module.MyService2` implements the `com.sun.star.lang.XInitialization` interface, so that `MyService2` can be instantiated with arguments passed to it during runtime.

```
#include <com/sun/star/uno/XInterface.idl>
```

```
#include <com/sun/star/lang/XInitialization.idl>

module my_module
{

interface XSomething : com::sun::star::uno::XInterface
{
    string methodOne( [in] string val );
};

service MyService1
{
    interface XSomething;
};

service MyService2
{
    interface XSomething;
    interface com::sun::star::lang::XInitialization;
};

};
```

This IDL is compiled to produce a binary type library file (*.urd* file), by executing the following commands. The types are compiled and merged into a registry *simple_component.rdb*, that will be linked into the OpenOffice.org installation later.

```
$ idlc -I<SDK>/idl some.idl
$ regmerge simple_component.rdb /UCR some.urd
```

The cppumaker tool must be used to map IDL to C++:

```
$ cppumaker -BUCR -Tmy_module.XSomething <SDK>/bin/applicat.rdb simple_component_rdb
```

For each given type, a pair of header files is generated, a *.hdl* and a *.hpp* file. To avoid conflicts, all C++ declarations of the type are in the *.hdl* and all definitions, such as constructors, are in the *.hpp* file. The *.hpp* is the one to include for any type used in C++.

The next step is to implement the core interfaces, and the implementation of the component operations `component_getFactory()`, `component_writeInfo()` and `component_getImplementationEnvironment()` with or without helper methods.

4.6.1 Class Definition with Helper Template Classes

XInterface, XTypeProvider and XWeak

The SDK offers helpers for ease of developing. There are implementation helper template classes that deal with the implementation of `com.sun.star.uno.XInterface` and `com.sun.star.lang.XTypeProvider`, as well as `com.sun.star.uno.XWeak`. These classes let you focus on the interfaces you want to implement.

The implementation of `my_module.MyService2` uses the `::cppu::WeakImplHelper3<>` helper. The “3” stands for the number of interfaces to implement. The class declaration inherits from this template class which takes the interfaces to implement as template parameters. (Components/CppComponent/service2_impl.cxx)

```
#include <cppuhelper/implbase3.hxx> // "3" implementing three interfaces
#include <cppuhelper/factory.hxx>
#include <cppuhelper/implementationentry.hxx>

#include <com/sun/star/lang/XServiceInfo.hpp>
#include <com/sun/star/lang/XInitialization.hpp>
#include <com/sun/star/lang/IllegalArgumentException.hpp>
#include <my_module/XSomething.hpp>


using namespace ::rtl; // for OUString
using namespace ::com::sun::star; // for sdk interfaces
using namespace ::com::sun::star::uno; // for basic types
```

```
namespace my_sc_impl {

class MyService2Impl : public ::cppu::WeakImplHelper3< ::my_module::XSomething,
                                                       lang::XServiceInfo,
                                                       lang::XInitialization >
{
    ...
};
}
```

The next section focusses on coding `com.sun.star.lang.XServiceInfo`, `com.sun.star.lang.XInitialization` and the sample interface my_module.XSomething.

The cppuhelper shared library provides additional implementation helper classes, for example, supporting `com.sun.star.lang.XComponent`. Browse the *::cppu* namespace in the C++ reference of the SDK or on *udk.openoffice.org*.

XServiceInfo

An UNO service implementation supports `com.sun.star.lang.XServiceInfo` providing information about its implementation name and supported services. The implementation name is a unique name referencing the specific implementation. In this case, `my_module.my_sc_impl.MyService1` and `my_module.my_sc_impl.MyService2` respectively. The implementation name is used later when registering the implementation into the *simple_component.rdb* registry used for OpenOffice.org. It links a service name entry to one implementation, because there may be more than one implementation. Multiple implementations of the same service may have different characteristics, such as runtime behavior and memory footprint.

Our service instance has to support the `com.sun.star.lang.XServiceInfo` interface. This interface has three methods, and can be coded for one supported service as follows: (Components/CppComponent/service2_impl.cxx)

```
// XServiceInfo implementation
OUString MyService2Impl::getImplementationName()
    throw (RuntimeException)
{
    // unique implementation name
    return OUString( RTL_CONSTASCII_USTRINGPARAM("my_module.my_sc_impl.MyService2") );
}
sal_Bool MyService2Impl::supportsService( OUString const & serviceName )
    throw (RuntimeException)
{
    // this object only supports one service, so the test is simple
    return serviceName.equalsAsciiL( RTL_CONSTASCII_STRINGPARAM("my_module.MyService2") );
}
Sequence< OUString > MyService2Impl::getSupportedServiceNames()
    throw (RuntimeException)
{
    return getSupportedServiceNames_MyService2Impl();
}
```

4.6.2 Implementing your own Interfaces

For the `my_module.XSomething` interface, add a string to be returned that informs the caller when `methodOne()` was called successfully . (Components/CppComponent/service2_impl.cxx)

```
OUString MyService2Impl::methodOne( OUString const & str )
    throw (RuntimeException)
{
    return OUString( RTL_CONSTASCII_USTRINGPARAM(
        "called methodOne() of MyService2 implementation: ") ) + str;
}
```

4.6.3 Providing a Single Factory Using a Helper Method

C++ component libraries must export an external "C" function called `component_getFactory()` that supplies a factory object for the given implementation. Use `::cppu::component_getFactoryHelper()` to create this function. The declarations for it are included through cppuhelper/implementationentry.hxx.

The `component_getFactory()` method appears at the end of the following listing. This method assumes that the component includes a static `::cppu::ImplementationEntry` array `s_component_entries[]`, which contains a number of function pointers. The listing shows how to write the component, so that the function pointers for all services of a multi-service component are correctly initialized. (Components/CppComponent/service2_impl.cxx)

```
#include <cppuhelper/implbase3.hxx> // "3" implementing three interfaces
#include <cppuhelper/factory.hxx>
#include <cppuhelper/implementationentry.hxx>

#include <com/sun/star/lang/XServiceInfo.hpp>
#include <com/sun/star/lang/XInitialization.hpp>
#include <com/sun/star/lang/IllegalArgumentException.hpp>
#include <my_module/XSomething.hpp>


using namespace ::rtl; // for OUString
using namespace ::com::sun::star; // for sdk interfaces
using namespace ::com::sun::star::uno; // for basic types


namespace my_sc_impl
{

class MyService2Impl : public ::cppu::WeakImplHelper3<
      ::my_module::XSomething, lang::XServiceInfo, lang::XInitialization >
{
    OUString m_arg;
public:
    // focus on three given interfaces,
    // no need to implement XInterface, XTypeProvider, XWeak

    // XInitialization will be called upon createInstanceWithArguments[AndContext]()
    virtual void SAL_CALL initialize( Sequence< Any > const & args )
        throw (Exception);
    // XSomething
    virtual OUString SAL_CALL methodOne( OUString const & str )
        throw (RuntimeException);
    // XServiceInfo
    virtual OUString SAL_CALL getImplementationName()
        throw (RuntimeException);
    virtual sal_Bool SAL_CALL supportsService( OUString const & serviceName )
        throw (RuntimeException);
    virtual Sequence< OUString > SAL_CALL getSupportedServiceNames()
        throw (RuntimeException);
};

// Implementation of XSomething, XServiceInfo and XInitilization omitted here:
...

// component operations from service1_impl.cxx
extern Sequence< OUString > SAL_CALL  getSupportedServiceNames_MyService1Impl();
extern OUString SAL_CALL getImplementationName_MyService1Impl();
extern Reference< XInterface > SAL_CALL create_MyService1Impl(
                                        Reference< XComponentContext > const & xContext )
                                            SAL_THROW( () );
// component operations for MyService2Impl
static Sequence< OUString > getSupportedServiceNames_MyService2Impl()
{
    static Sequence < OUString > *pNames = 0;
    if( ! pNames )
    {
        if( !pNames )
        {
            static Sequence< OUString > seqNames(1);
            seqNames.getArray()[0] = OUString(RTL_CONSTASCII_USTRINGPARAM("my_module.MyService2"));
            pNames = &seqNames;
        }
    }
    return *pNames;
}

static OUString getImplementationName_MyService2Impl()
```

```
{
    static OUString *pImplName = 0;
    if( ! pImplName )
    {
        if( ! pImplName )
        {
            static OUString implName(
                RTL_CONSTASCII_USTRINGPARAM("my_module.my_sc_implementation.MyService2") );
            pImplName = &implName;
        }
    }
    return *pImplName;
}


Reference< XInterface > SAL_CALL create_MyService2Impl(
    Reference< XComponentContext > const & xContext )
    SAL_THROW( () )
{
    return static_cast< lang::XTypeProvider * >( new MyService2Impl() );
}


/* shared lib exports implemented with helpers */
static struct ::cppu::ImplementationEntry s_component_entries [] =
{
    {
        create_MyService1Impl, getImplementationName_MyService1Impl,
        getSupportedServiceNames_MyService1Impl, ::cppu::createSingleComponentFactory,
        0, 0
    },
    {
        create_MyService2Impl, getImplementationName_MyService2Impl,
        getSupportedServiceNames_MyService2Impl, ::cppu::createSingleComponentFactory,
        0, 0
    },
    { 0, 0, 0, 0, 0, 0 }
};
}

extern "C"
{
void * SAL_CALL component_getFactory(
    sal_Char const * implName, lang::XMultiServiceFactory * xMgr,
    registry::XRegistryKey * xRegistry )
{
    return ::cppu::component_getFactoryHelper(
        implName, xMgr, xRegistry, ::my_sc_impl::s_component_entries );
}

//  getImplementationEnvironment and component_writeInfo are described later, we omit them here
...
}
```

The static variable `s_component_entries` defines a null-terminated array of entries concerning the service implementations of the shared library. A service implementation entry consists of function pointers for

- object creation: `create_MyServiceXImpl()`
- implementation name: `getImplementationName_MyServiceXImpl()`
- supported service names: `getSupportedServiceNames_MyServiceXImpl()`
- factory helper to be used: `::cppu::createComponentFactory()`

The last two values are reserved for future use and therefore can be 0.

4.6.4 Write Registration Info Using Helper Method

Use `::cppu::component_writeInfoHelper()` to implement `component_writeInfo()`: This function is called by *regcomp* during the registration process.
[ScOURCE:Components/simple_cpp_component/service2_impl.cxx]

```
extern "C" sal_Bool SAL_CALL component_writeInfo(
    lang::XMultiServiceFactory * xMgr, registry::XRegistryKey * xRegistry )
{
    return ::cppu::component_writeInfoHelper(
        xMgr, xRegistry, ::my_sc_impl::s_component_entries );
}
```

Note that `component_writeInfoHelper()` uses the same array of `::cppu::ImplementationEntry` structs as `component_getFactory()`,that is, `s_component_entries`.

4.6.5 Provide Implementation Environment

The function called `component_getImplementationEnvironment()` tells the shared library component loader which compiler was used to build the library. This information is required if different components have been compiled with different compilers. A specific C++-compiler is called an environment. If different compilers were used, the loader has to bridge interfaces from one compiler environment to another, building the infrastructure of communication between those objects. It is mandatory to have the appropriate C++ bridges installed into the UNO runtime. In most cases, the function mentioned above can be implemented this way: (Components/CppComponent/service2_impl.cxx)

```
extern "C" void SAL_CALL component_getImplementationEnvironment(
    sal_Char const ** ppEnvTypeName, uno_Environment ** ppEnv )
{
    *ppEnvTypeName = CPPU_CURRENT_LANGUAGE_BINDING_NAME;
}
```

The macro `CPPU_CURRENT_LANGUAGE_BINDING_NAME` is a C string defined by the compiling environment, if you use the SDK compiling environment. For example, when compiling with the Microsoft Visual C++ compiler, it defines to `"msci"`, but when compiling with the GNU gcc 3, it defines to `"gcc3"`.

4.6.6 Implementing without Helpers

In the following section, possible implementations without helpers are presented. This is useful if more interfaces are to be implemented than planned by the helper templates. The helper templates only allow up to ten interfaces. Also included in this section is how the core interfaces work.

XInterface Implementation

Object lifetime is controlled through the common base interface `com.sun.star.uno.XInterface` methods `acquire()` and `release()`. These are implemented using reference-counting, that is, upon each `acquire()`, the counter is incremented and upon each `release()`, it is decreased. On last decrement, the object dies. Programming in a thread-safe manner, the modification of this counter member variable is commonly performed by a pair of `sal` library functions called `osl_incrementInterlockedcount()` and `osl_decrementInterlockedcount()` (include *osl/interlck.h*). (Components/CppComponent/service1_impl.cxx)



Be aware of symbol conflicts when writing code. It is common practice to wrap code into a separate namespace, such as `"my_sc_impl"`. The problem is that symbols may clash during runtime on Unix when your shared library is loaded.

```
namespace my_sc_impl
{
class MyService1Impl
    ...
{
```

```
    oslInterlockedCount m_refcount;
public:
    inline MyService1Impl() throw ()
        : m_refcount( 0 )
        {}

    // XInterface
    virtual Any SAL_CALL queryInterface( Type const & type )
        throw (RuntimeException);
    virtual void SAL_CALL acquire()
        throw ();
    virtual void SAL_CALL release()
        throw ();
    ...
};
void MyService1Impl::acquire()
    throw ()
{
    // thread-safe incrementation of reference count
    ::osl_incrementInterlockedCount( &m_refcount );
}
void MyService1Impl::release()
    throw ()
{
    // thread-safe decrementation of reference count
    if (0 == ::osl_decrementInterlockedCount( &m_refcount ))
    {
        delete this; // shutdown this object
    }
}
```

In the `queryInterface()` method, interface pointers have to be provided to the interfaces of the object. That means, cast `this` to the respective pure virtual C++ class generated by the *cppumaker* tool for the interfaces. All supported interfaces must be returned, including inherited interfaces like `XInterface`. (Components/CppComponent/service1_impl.cxx)

```
Any MyService1Impl::queryInterface( Type const & type )
    throw (RuntimeException)
{
    if (type.equals( ::getCppuType( (Reference< XInterface > const *)0 ) ))
    {
        // return XInterface interface (resolve ambiguity caused by multiple inheritance from
        // XInterface subclasses by casting to lang::XTypeProvider)
        Reference< XInterface > x( static_cast< lang::XTypeProvider * >( this ) );
        return makeAny( x );
    }
    if (type.equals( ::getCppuType( (Reference< lang::XTypeProvider > const *)0 ) ))
    {
        // return XInterface interface
        Reference< XInterface > x( static_cast< lang::XTypeProvider * >( this ) );
        return makeAny( x );
    }
    if (type.equals( ::getCppuType( (Reference< lang::XServiceInfo > const *)0 ) ))
    {
        // return XServiceInfo interface
        Reference< lang::XServiceInfo > x( static_cast< lang::XServiceInfo * >( this ) );
        return makeAny( x );
    }
    if (type.equals( ::getCppuType( (Reference< ::my_module::XSomething > const *)0 ) ))
    {
        // return sample interface
        Reference< ::my_module::XSomething > x( static_cast< ::my_module::XSomething * >( this ) );
        return makeAny( x );
    }
    // querying for unsupported type
    return Any();
}
```

XTypeProvider Implementation

When implementing the `com.sun.star.lang.XTypeProvider` interface, two methods have to be coded. The first one, `getTypes()` provides all implemented types of the implementation, excluding base types, such as `com.sun.star.uno.XInterface`. The second one, `getImplementationId()` provides a unique ID for this set of interfaces. A thread-safe implementation of the above mentioned looks like the following example:
(Components/CppComponent/service1_impl.cxx)

```
Sequence< Type > MyService1Impl::getTypes()
```

```
    throw (RuntimeException)
{
    Sequence< Type > seq( 3 );
    seq[ 0 ] = ::getCppuType( (Reference< lang::XTypeProvider > const *)0 );
    seq[ 1 ] = ::getCppuType( (Reference< lang::XServiceInfo > const *)0 );
    seq[ 2 ] = ::getCppuType( (Reference< ::my_module::XSomething > const *)0 );
    return seq;
}
Sequence< sal_Int8 > MyService1Impl::getImplementationId()
    throw (RuntimeException)
{
    static Sequence< sal_Int8 > * s_pId = 0;
    if (! s_pId)
    {
        // create unique id
        Sequence< sal_Int8 > id( 16 );
                ::rtl_createUuid( (sal_uInt8 *)id.getArray(), 0, sal_True );
        // guard initialization with some mutex
        ::osl::MutexGuard guard( ::osl::Mutex::getGlobalMutex() );
        if (! s_pId)
        {
            static Sequence< sal_Int8 > s_id( id );
            s_pId = &s_id;
        }
    }
    return *s_pId;
}
```



Take a look at the thread-safe initialization of the implementation ID. A common pattern is to test a static pointer that is modified by one atom write. Using the same pattern, you can do a static initialization of the types sequence. This has been omitted for simplicity.
In general, do not `acquire()` mutexes when calling alien code if you do not know what the called code is doing. You never know what mutexes the alien code is acquiring which can lead to deadlocks. This is the reason, why the latter value (uuid) is created before the initialization mutex is acquired. After the mutex is successfully acquired, the value of s_pID is checked again and assigned if it has not been assigned before. This is the design pattern double check lock.

Providing a Single Factory

The function `component_getFactory()` provides a single object factory for the requested implementation, that is, it provides a factory that creates object instances of one of the service implementations. Using a helper from *cppuhelper/factory.hxx*, this is implemented quickly in the following code: (Components/CppComponent/service1_impl.cxx)

```
#include <cppuhelper/factory.hxx>

namespace my_sc_impl
{
...
static Reference< XInterface > SAL_CALL create_MyService1Impl(
    Reference< XComponentContext > const & xContext )
    SAL_THROW( () )
{
    return static_cast< lang::XTypeProvider * >( new MyService1Impl() );
}
static Reference< XInterface > SAL_CALL create_MyService2Impl(
    Reference< XComponentContext > const & xContext )
    SAL_THROW( () )
{
    return static_cast< lang::XTypeProvider * >( new MyService2Impl() );
}
}

extern "C" void * SAL_CALL component_getFactory(
    sal_Char const * implName, lang::XMultiServiceFactory * xMgr, void * )
{
    Reference< lang::XSingleComponentFactory > xFactory;
    if (0 == ::rtl_str_compare( implName, "my_module.my_sc_impl.MyService1" ))
    {
        // create component factory for MyService1 implementation
        OUString serviceName( RTL_CONSTASCII_USTRINGPARAM("my_module.MyService1") );
        xFactory = ::cppu::createSingleComponentFactory(
            ::my_sc_impl::create_MyService1Impl,
            OUString( RTL_CONSTASCII_USTRINGPARAM("my_module.my_sc_impl.MyService1") ),
```

```
            Sequence< OUString >( &serviceName, 1 ) );
    }
    else if (0 == ::rtl_str_compare( implName, "my_module.my_sc_impl.MyService2" ))
    {
        // create component factory for MyService12 implementation
        OUString serviceName( RTL_CONSTASCII_USTRINGPARAM("my_module.MyService2") );
        xFactory = ::cppu::createSingleComponentFactory(
            ::my_sc_impl::create_MyService2Impl,
            OUString( RTL_CONSTASCII_USTRINGPARAM("my_module.my_sc_impl.MyService2") ),
            Sequence< OUString >( &serviceName, 1 ) );
    }
    if (xFactory.is())
        xFactory->acquire();
    return xFactory.get(); // return acquired interface pointer or null
}
```

In the example above, note the function `::my_sc_impl::create_MyService1Impl()` that is called by the factory object when it needs to instantiate the class. A component context `com.sun.star.uno.XComponentContext` is provided to the function, which may be passed to the constructor of `MyService1Impl`.

Write Registration Info

The function `component_writeInfo()` is called by the shared library component loader upon registering the component into a registry database file (*.rdb*). The component writes information about objects it can instantiate into the registry when it is called by *regcomp*. (Components/CppComponent/service1_impl.cxx)

```
extern "C" sal_Bool SAL_CALL component_writeInfo(
    lang::XMultiServiceFactory * xMgr, registry::XRegistryKey * xRegistry )
{
    if (xRegistry)
    {
        try
        {
            // implementation of MyService1A
            Reference< registry::XRegistryKey > xKey(
                xRegistry->createKey( OUString( RTL_CONSTASCII_USTRINGPARAM(
                    "my_module.my_sc_impl.MyService1/UNO/SERVICES") ) ) );
            // subkeys denote implemented services of implementation
            xKey->createKey( OUString( RTL_CONSTASCII_USTRINGPARAM(
                "my_module.MyService1") ) );
            // implementation of MyService1B
            xKey = xRegistry->createKey( OUString( RTL_CONSTASCII_USTRINGPARAM(
                "my_module.my_sc_impl.MyService2/UNO/SERVICES") ) );
            // subkeys denote implemented services of implementation
            xKey->createKey( OUString( RTL_CONSTASCII_USTRINGPARAM(
                "my_module.MyService2") ) );
            return sal_True; // success
        }
        catch (registry::InvalidRegistryException &)
        {
            // function fails if exception caught
        }
    }
    return sal_False;
}
```

4.6.7 Storing the Service Manager for Further Use

The single factories expect a static `create_<ImplementationClass>()` function. For instance, `create_MyService1Impl()` takes a reference to the component context and instantiates the implementation class using `new ImplementationClass()`. A constructor can be written for `<ImplementationClass>` that expects a reference to an `com.sun.star.uno.XComponentContext` and stores the reference in the instance for further use.

```
static Reference< XInterface > SAL_CALL create_MyService2Impl(
    Reference< XComponentContext > const & xContext )
    SAL_THROW( () )
{
    // passing the component context to the constructor of MyService2Impl
```

```
    return static_cast< lang::XTypeProvider * >( new MyService2Impl( xContext ) );
}
```

4.6.8 Create Instance with Arguments

If the service should be raised passing arguments through `com.sun.star.lang.XMultiComponentFactory:createInstanceWithArgumentsAndContext()` and `com.sun.star.lang.XMultiServiceFactory:createInstanceWithArguments()`, it has to implement the interface `com.sun.star.lang.XInitialization`. The second service `my_module.MyService2` implements it, expecting a single string as an argument. (Components/CppComponent/service2_impl.cxx)

```
// XInitialization implementation
void MyService2Impl::initialize( Sequence< Any > const & args )
    throw (Exception)
{
    if (1 != args.getLength())
    {
        throw lang::IllegalArgumentException(
            OUString( RTL_CONSTASCII_USTRINGPARAM("give a string instanciating this component!") ),
            (::cppu::OWeakObject *)this, // resolve to XInterface reference
            0 ); // argument pos
    }
    if (! (args[ 0 ] >>= m_arg))
    {
        throw lang::IllegalArgumentException(
            OUString( RTL_CONSTASCII_USTRINGPARAM("no string given as argument!") ),
            (::cppu::OWeakObject *)this, // resolve to XInterface reference
            0 ); // argument pos
    }
}
```

4.6.9 Multiple Components in One Dynamic Link Library

The construction of C++ components allows putting as many service implementations into a component file as desired. Ensure that the component operations are implemented in such a way that `component_writeInfo()` and `component_getFactory()` handle all services correctly. Refer to the sample component `simple_component` to see an example on how to implement two services in one link library.

4.6.10 Building and Testing C++ Components

Build Process

For details about building component code, see the gnu makefile. It uses a number of platform dependent variables used in the SDK that are included from *<SDK>/settings/settings.mk*. For simplicity, details are omitted here, and the build process is just sketched in eight steps:

1. The UNOIDL compiler compiles the .idl file *some.idl* into an urd file.
2. The resulting binary *.urd* files are merged into a new *simple_component.rdb*.
3. The tool *xml2cmp* parses the xml component description *simple_component.xml* for types needed for compiling. This file describes the service implementation(s) for deployment, such as the purpose of the implementation(s) and used types. Visit http://udk.openoffice.org/common/man/module_description.html for details about the syntax of these XML files.

4. The types parsed in step 3 are passed to *cppumaker*, which generates the appropriate header pairs into the output include directory using *simple_component.rdb* and the OpenOffice.org type library *applicat.rdb* that is stored in the program directory of your OpenOffice.org installation.



For your own component you can simplify step 3 and 4, and pass the types used by your component to cppumaker using the -T option.

5. The source files *service1_impl.cxx* and *service2_impl.cxx* are compiled.
6. The shared library is linked out of object files, linking dynamically to the UNO base libraries sal, cppu and cppuhelper. The shared library's name is *libsimple_component.so* on Unix and *simple_component.dll* on Windows.



In general, the shared library component should limit its exports to only the above mentioned functions (prefixed with component_) to avoid symbol clashes on Unix. In addition, for the gnu gcc3 C++ compiler, it is necessary to export the RTTI symbols of exceptions, too.

7. The shared library component is registered into *simple_component.rdb*. This can also be done manually running

```
$ regcomp -register -r simple_component.rdb -c simple_component.dll
```

Test Registration and Use

The component's registry *simple_component.rdb* has entries for the registered service implementations. If the library is registered successfully, run:

```
$ regview simple_component.rdb
```

The result should look similar to the following:

```
/
 / UCR
   / my_module
     / XSomething

       ... interface information ...

 / IMPLEMENTATIONS
   / my_module.my_sc_impl.MyService2
     / UNO
       / ACTIVATOR
         Value: Type = RG_VALUETYPE_STRING
                Size = 34
                Data = "com.sun.star.loader.SharedLibrary"

       / SERVICES
         / my_module.MyService2
       / LOCATION
         Value: Type = RG_VALUETYPE_STRING
                Size = 21
                Data = "simple_component.dll"

   / my_module.my_sc_impl.MyService1
     / UNO
       / ACTIVATOR
         Value: Type = RG_VALUETYPE_STRING
                Size = 34
                Data = "com.sun.star.loader.SharedLibrary"

       / SERVICES
         / my_module.MyService1
       / LOCATION
         Value: Type = RG_VALUETYPE_STRING
                Size = 21
                Data = "simple_component.dll"

 / SERVICES
   / my_module.MyService1
     Value: Type = RG_VALUETYPE_STRINGLIST
            Size = 40
            Len  = 1
```

```
            Data = 0 = "my_module.my_sc_impl.MyService1"

   / my_module.MyService2
     Value: Type = RG_VALUETYPE_STRINGLIST
            Size = 40
            Len  = 1
            Data = 0 = "my_module.my_sc_impl.MyService2"
```

OpenOffice.org recognizes registry files being inserted into the *unorc* file (on Unix, *uno.ini* on Windows) in the program directory of your OpenOffice.org installation. Extend the types and services in that file by *simple_component.rdb*. The given file has to be an absolute file URL, but if the rdb is copied to the OpenOffice.org program directory, a $SYSBINDIR macro can be used, as shown in the following *unorc* file:

```
[Bootstrap]
UNO_TYPES=$SYSBINDIR/applicat.rdb $SYSBINDIR/simple_component.rdb
UNO_SERVICES=$SYSBINDIR/applicat.rdb $SYSBINDIR/simple_component.rdb
```

Second, when running OpenOffice.org, extend the PATH (Windows) or LD_LIBRARY_PATH (Unix), including the output path of the build, so that the loader finds the component. If the shared library is copied to the program directory or a link is created inside the program directory (Unix only), do not extend the path.

Launching the test component inside a OpenOffice.org Basic script is simple to do, as shown in the following code:

```
Sub Main

    REM calling service1 impl
    mgr = getProcessServiceManager()
    o = mgr.createInstance("my_module.MyService1")
    MsgBox o.methodOne("foo")
    MsgBox o.dbg_supportedInterfaces

    REM calling service2 impl
    dim args( 0 )
    args( 0 ) = "foo"
    o = mgr.createInstanceWithArguments("my_module.MyService2", args())
    MsgBox o.methodOne("bar")
    MsgBox o.dbg_supportedInterfaces

End Sub
```

This procedure instantiates the service implementations and performs calls on their interfaces. The return value of the `methodOne()` call is brought up in message boxes. The Basic object property `dbg_supportedInterfaces` retrieves its information through the `com.sun.star.lang.XTypeProvider` interfaces of the objects.

4.7 Integrating Components into OpenOffice.org

If a component needs to be called from the OpenOffice.org user interface, it must be able to take part in the communication between the UI layer and the application objects. OpenOffice.org uses command URLs for this purpose. When a user chooses an item in the user interface, a command URL is dispatched to the application framework and processed in a chain of responsibility until an object accepts the command and executes it, thus consuming the command URL. This mechanism is known as the *dispatch framework*, it is covered in detail in chapter *6.1.6 Office Development - OpenOffice.org Application Environment - Using the Dispatch Framework*.

From version 1.1, OpenOffice.org provides user interface support for custom components by two basic mechanisms:

- Components can be enabled to process command URLs. There are two ways to accomplish this. You can either make them a *protocol handler* for command URLs or integrate them into the *job execution environment* of OpenOffice.org. The protocol handler technique is simple, but it can only be used with command URLs in the dispatch framework. A component for the job execu-

tion environment can be used with or without command URLs, and has comprehensive support when it comes to configuration, job environment, and lifetime issues.

- The user interface can be adjusted to new components. On the one hand, you can add new menus and toolbar items and configure them to send the command URLs needed for your component. On the other hand, it is possible to disable existing commands. All this is possible by adding certain files to the UNO package distribution. When users of your component deploy the package into an individual or a network OpenOffice.org installation, the GUI is adjusted automatically.

The left side of Illustration 35 shows the two possibilities for processing command URLs: either custom protocol handlers or the specialized job protocol. On the right, you see the job execution environment, which is used by the job protocol, but can also be used without command URLs from any source code.

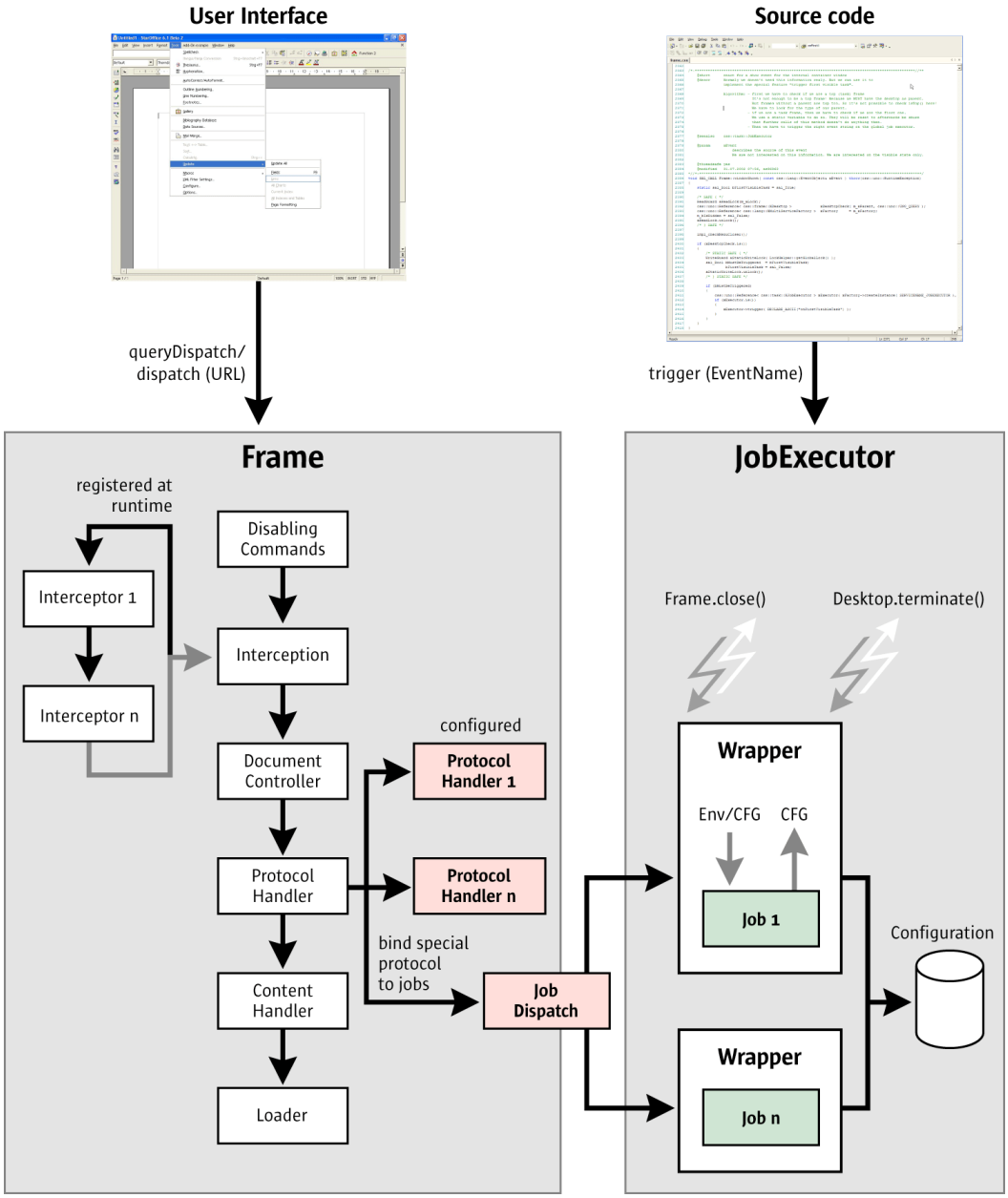


Illustration 35: Processing command URLs and the job execution environment

This section describes how to use these mechanisms. It discusses protocol handlers and jobs, then describes how to customize the OpenOffice.org user interface for components.

4.7.1 Protocol Handler

The dispatch framework binds user interface controls, such as menu or toolbar items, to the functionality of OpenOffice.org. Every function that is reachable in the user interface is described by a command URL and corresponding parameters.

The protocol handler mechanism is an API that enables programmers to add arbitrary URL schemas to the existing set of command URLs by writing additional protocol handlers for them. Such a protocol handler must be implemented as a UNO component and registered in the OpenOffice.org configuration for the new URL schema.

Overview

To issue a command URL, the first step is to locate a dispatch object that is responsible for the URL. Start with the frame that contains the document for which the command is meant. Its interface method `com.sun.star.frame.XDispatchProvider:queryDispatch()` is called with a URL and special search parameters to locate the correct target. This request is passed through the following instances:

disabling commands	Checks if command is on the list of disabled commands, described in *4.7.4 Writing UNO Components - Integrating Components into OpenOffice.org - Disable Commands*
interception	Intercepts command and re-routes it, described in *6.1.6 Office Development - OpenOffice.org Application Environment - Using the Dispatch Framework - Dispatch Interception*
targeting	Determines target frame for command, described in *6.1.5 Office Development - OpenOffice.org Application Environment - Handling Documents - Loading Documents - Target Frame*
controller	Lets the controller of the frame try to handle the command, described in *6.1.6 Office Development - OpenOffice.org Application Environment - Using the Dispatch Framework - Processing Chain*
protocol handler	Determines if there is a custom handler for the command, described in this section
interpret as loadable content	Loads content from file, described in *6.1.5 Office Development - OpenOffice.org Application Environment - Handling Documents - Loading Documents - URL Parameter*. Generally contents are loaded into a frame by a `com.sun.star.frame.FrameLoader`, but if a content (e.g. a sound) needs no frame, a `com.sun.star.frame.ContentHandler` service is used, which needs no target frame for its operation.

The list shows that the protocol handler will only be used if the URL has not been called before. Because targeting has already been done, it is clear that the command will run in the located target frame environment, which is usually `"_self"`.



The target "_blank" cannot be used for a protocol handler. Since "_blank" leads to the creation of a new frame for a component, there would be no component yet for the protocol handler to work with.

A protocol handler decides by itself if it returns a valid dispatch object, that is, it is asked to agree with the given request by the dispatch framework. If a dispatch object is returned, the requester can use it to dispatch the URL by calling its `dispatch()` method.

Implementation

A protocol handler implementation must follow the service definition `com.sun.star.frame.ProtocolHandler`. At least the interface `com.sun.star.frame.XDispatchProvider` must be supported.

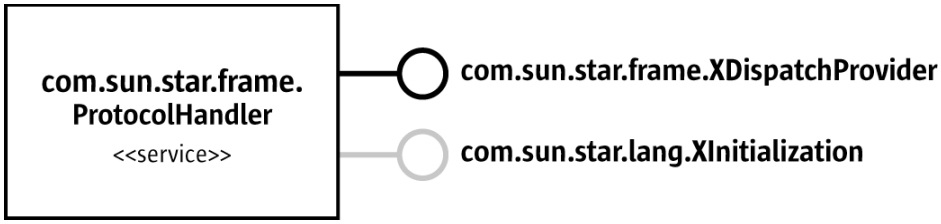


Illustration 36: Protocol handler

The interface `XDispatchProvider` supports two methods:

```
XDispatch queryDispatch(  [in] ::com::sun::star::util::URL URL,
                          [in] string  TargetFrameName,
                          [in] long SearchFlags )
sequence< XDispatch > queryDispatches(  [in] sequence< DispatchDescriptor >  Requests )
```

The protocol handler is asked for its agreement to execute a given URL by a call to the interface method `com.sun.star.frame.XDispatchProvider:queryDispatch()`. The incoming URL should be parsed and validated. If the URL is valid and the protocol handler is able to handle it, it should return a dispatch object, thus indicating that it accepts the request.

The dispatch object must support the interface `com.sun.star.frame.XDispatch` with the methods

```
[oneway] void dispatch(  [in] ::com::sun::star::util::URL URL,
                         [in] sequence< ::com::sun::star::beans::PropertyValue >  Arguments )
addStatusListener [oneway] void addStatusListener(  [in] XStatusListener Control,
                                                    [in] ::com::sun::star::util::URL  URL )
removeStatusListener [oneway] void removeStatusListener(  [in] XStatusListener Control,
                                                          [in] ::com::sun::star::util::URL  URL )
```

Optionally, the dispatch object can support the interface `com.sun.star.frame.XNotifyingDispatch`, which derives from `XDispatch` and introduces a new method `dispatchWithNotification()`. This interface is preferred if it is present.

```
[oneway] void dispatchWithNotification(
        [in] com::sun::star::util::URL URL,
        [in] sequence<com::sun::star::beans::PropertyValue> Arguments,
        [in] com::sun::star::frame::XDispatchResultListener Listener);
```

A basic protocol handler is free to implement `XDispatch` itself, so it can simply return itself in the `queryDispatch()` implementation. But it is advisable to return specialized helper dispatch objects instead of the protocol handler instance. This helps to decrease the complexity of status updates. It is easier to notify status listeners for a single-use dispatch object instead of multi-use dispatch objects, which have to distinguish the URLs given in `addStatusListener()` all the time.



To supply the UI with status information for a command, it is required to call back a `com.sun.star.frame.XStatusListener` during its registration immediately, for example:

```
public void addStatusListener(XStatusListener xControl, URL aURL) {
    FeatureStateEvent aState = new FeatureStateEvent();
    aState.FeatureURL = aURL;
    aState.IsEnabled  = true;
    aState.State      = Boolean.TRUE;
    xControl.statusChanged(aState);
    m_lListenerContainer.add(xControl);
}
```

A protocol handler can support the interface `com.sun.star.lang.XInitialization` if it wants to be initialized with a `com.sun.star.frame.Frame` environment to work with. `XInitialization` contains one method:

```
void initialize(  [in] sequence< any > aArguments )
```

A protocol handler is generally used in a well known `com.sun.star.frame.Frame` context, therefore the dispatch framework always passes this frame context through `initialize()` as the first argument, if `XInitialization` is present. Its `com.sun.star.frame.XFrame` interface provides access to the controller, from which you can get the document model and have a good starting point to work with the document.

Illustration 36 shows how to get to the controller and the document model from an `XFrame` interface. The chapter *6.1.3 Office Development - OpenOffice.org Application Environment - Using the Component Framework* describes the usage of frames, controllers and models in more detail.

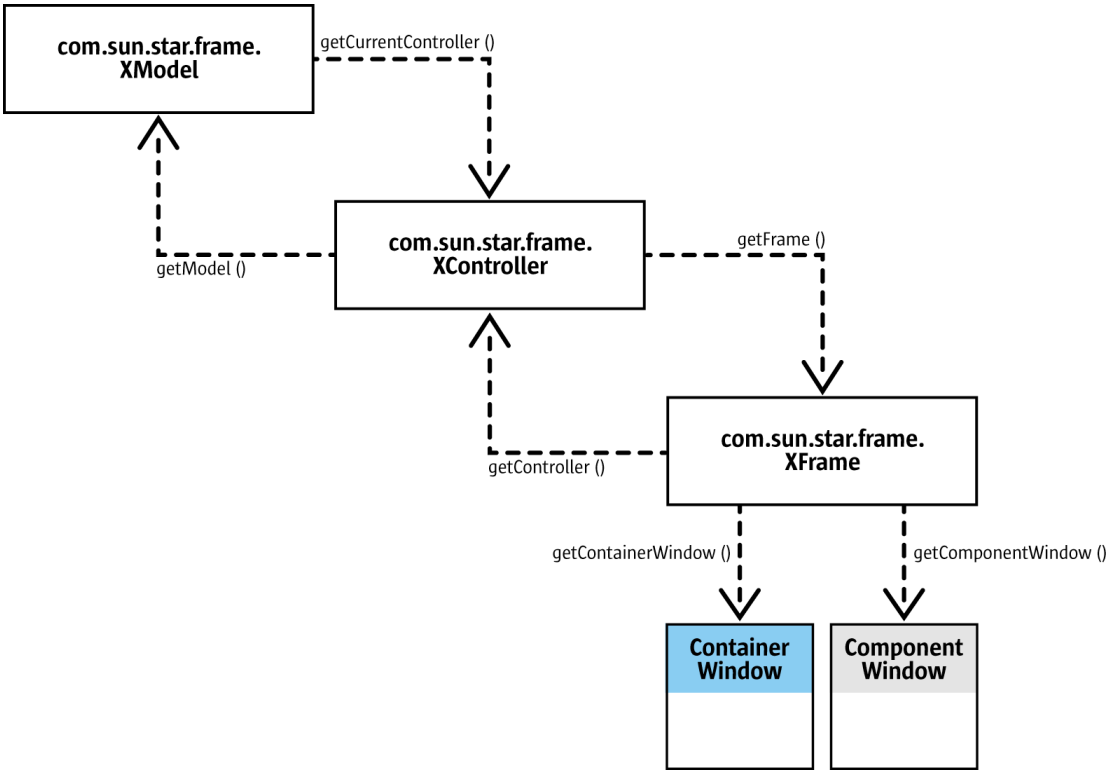


Illustration 37: Frame-controller-model organization



A protocol handler can be implemented as a singleton, but this poses multithreading difficulties. In a multi-threaded environment it is most unlikely that the initial frame context matches every following dispatch request. So you have to be prepared for calls to `initialize()` by multiple threads for multiple frames. A dispatch object can also be used more then once, but must be bound to the target frame that was specified in the original `queryDispatch()` call. A change of the frame context can cause trouble if the protocol handler returns itself as a dispatch object. A protocol handler singleton must return new dispatch objects for every request, which has to be initialized with the current context of the protocol handler, and you have to synchronize between `initialize()` and `queryDispatch()`. The protocol handler would have to serve as a kind of factory for specialized dispatch objects.
You can avoid these problems, if you write your protocol handler as a multi-instance service.

The opportunity to deny a `queryDispatch()` call allows you to register a protocol handler for a URL schema using wildcards, and to accept only a subset of all possible URLs. That way the handler object can validate incoming URLs and reject them if they appear to be invalid. However, this feature should not be used to register different protocol handlers for the same URL schema and accept different subsets by different handler objects, because it would be very difficult to avoid ambiguities.

Since a protocol handler is a UNO component, it must contain the component operations needed by a UNO service manager. These operations are certain static methods in Java or export functions in C++. It also has to implement the core interfaces used to enable communication with UNO and the application environment. For more information on the component operations and core interfaces, please see *4.3 Writing UNO Components - Component Architecture* and *4.4 Writing UNO Components - Core Interfaces to Implement.*

Java Protocol Handler - vnd.sun.star.framework.ExampleHandler

The following example shows a simple protocol handler implementation in Java. For simplicity, the component operations are omitted.

```
// imports
#import com.sun.star.beans.*;
#import com.sun.star.frame.*;
#import com.sun.star.uno.*;
#import com.sun.star.util.*;

// definition
public class ExampleHandler implements com.sun.star.frame.XDispatchProvider,
        com.sun.star.lang.XInitialization {
    // member
    /** points to the frame context in which this handler runs, is set in initialize()*/
    private com.sun.star.frame.XFrame m_xContext;

    // Dispatch object as inner class
    class OwnDispatch implements com.sun.star.frame.XDispatch {
        /** the target frame, in which context this dispatch must work */
        private com.sun.star.frame.XFrame m_xContext;

        /** describe the function of this dispatch.
         *  Because a URL can contain e.g. optional arguments
         *  this URL means the main part of such URL sets only. */
        private com.sun.star.util.URL m_aMainURL;

        /** contains all interested status listener for this dispatch */
        private java.lang.HashMap m_lListener;

        /** take over all neccessary parameters from outside. */
        public OwnDispatch(com.sun.star.frame.XFrame xContext, com.sun.star.util.URL aMainURL) {
            m_xContext = xContext;
            m_aMainURL = aMainURL;
        }

        /** execute the functionality, which is described by this URL.
         *
         *  @param  aURL
         *          this URL can describe the main function, we already know;
         *          but it can specify a sub function too! But queryDispatch()
         *          and dispatch() are used in a generic way ...
         *          m_aMainURL and aURL will be the same.
         *
         *  @param  lArgs
```

```
         *          optional arguments for this request
         */
        public void dispatch(com.sun.star.util.URL aURL, com.sun.star.beans.PropertyValue lArgs)
            throws com.sun.star.uno.RuntimeException {
            // ... do function
            // ... inform listener if neccessary
        }

        /** register a new listener and bind it toe given URL.
         *
         *  Note: Because the listener does not know the current state
         *  and may nobody change it next time, it is neccessary to inform it
         *  immediatly about this current state. So the listener is up to date.
         */
        public void addStatusListener(com.sun.star.frame.XStatusListener xListener,
                com.sun.star.util.URL aURL) throws com.sun.star.uno.RuntimeException {
            // ... register listener for given URL
            // ... inform it immediatly about current state!
            xListener.statusChanged(...);
        }

        /** deregister a listener for this URL. */
        public void removeStatusListener(com.sun.star.frame.XStatusListener xListener,
                com.sun.star.util.URLaURL) throws com.sun.star.uno.RuntimeException {
            // ... deregister listener for given URL
        }
    }

    /** set the target frame reference as context for all following dispatches. */
    public void initialize(com.sun.star.uno.Any[] lContext) {
        m_xContext = (com.sun.star.frame.XFrame)com.sun.star.uno.AnyConverter.toObject(lContext[0]);
    }

    /** should return a valid dispatch object for the given URL.
     *
     *  In case the URL is not valid an empty reference can be returned.
     *  The parameter sTarget and nFlags can be ignored. The will be "_self" and 0
     *  everytime.
     */
    public com.sun.star.frame.XDispatch queryDispatch(com.sun.star.util.URL aURL,
            java.lang.String sTarget, int nFlags ) throws com.sun.star.uno.RuntimeException {
        // check if given URL is valid for this protocol handler
        if (!aURL.Main.startsWith("myProtocol_1://") && !aURL.Main.startsWith("myProtocol_2://"))
            return null;
        // and return a specialized dispatch object
        // Of course "return this" would be possible too ...
        return (com.sun.star.frame.XDispatch)(new OwnDispatch(m_xContext, aURL));
    }

    /** optimized API call for remote.
     *
     *  It should be forwarded to queryDispatch() for every request item of the
     *  given DispatchDescriptor list.
     *
     *  But note: it is not allowed to pack the return list of dispatch objects.
     *  Every request in source list must match to a reference (null or valid) in
     *  the destination list!
     */
    public com.sun.star.frame.XDispatch[] queryDispatches(
            com.sun.star.frame.DispatchDescriptor[] lRequests) throws com.sun.star.uno.RuntimeException
{
        int c = lRequests.length;
        com.sun.star.frame.XDispatch[] lDispatches = new com.sun.star.frame.XDispatch[c];
        for (int i=0; i<c; ++i)
            lDispatches[i] = queryDispatch(lRequests[i].FeatureURL,
                lRequests[i].FrameName, lRequests[i].SearchFlags);
        return lDispatches;
    }
}
```

C++ Protocol Handler - org.openoffice.Office.addon.example

The next example shows a protocol handler in C++. The section *4.7.3 Writing UNO Components - Integrating Components into OpenOffice.org - User Interface Add-Ons* below will integrate this example handler into the graphical user interface of OpenOffice.org.

The following code shows the UNO component operations that must be implemented in a C++ protocol handler example. The three C functions return vital information to the UNO environment:

- `component_getImplementationEnvironment()` tells the shared library component loader which compiler was used to build the library.
- `component_writeInfo()` is called during the registration process by the registration tool *regcomp*, or indirectly when you apply *pkgchk*
- `component_getFactory()` provides a single service factory for the requested implementation. This factory can be asked to create an arbitrary number of instances for only one service specification, therefore it is called a single service factory, as opposed to a multi-service factory, where you can order instances for many different service specifications. (A single service factory has nothing to do with a singleton).

```
#include <stdio.h>

#ifndef _RTL_USTRING_HXX_
#include <rtl/ustring.hxx>
#endif

#ifndef _CPPUHELPER_QUERYINTERFACE_HXX_
#include <cppuhelper/queryinterface.hxx> // helper for queryInterface() impl
#endif
#ifndef _CPPUHELPER_FACTORY_HXX_
#include <cppuhelper/factory.hxx> // helper for component factory
#endif
// generated c++ interfaces

#ifndef _COM_SUN_STAR_LANG_XSINGLESERVICEFACTORY_HPP_
#include <com/sun/star/lang/XSingleServiceFactory.hpp>
#endif
#ifndef _COM_SUN_STAR_LANG_XMULTISERVICEFACTORY_HPP_
#include <com/sun/star/lang/XMultiServiceFactory.hpp>
#endif
#ifndef _COM_SUN_STAR_LANG_XSERVICEINFO_HPP_
#include <com/sun/star/lang/XServiceInfo.hpp>
#endif
#ifndef _COM_SUN_STAR_REGISTRY_XREGISTRYKEY_HPP_
#include <com/sun/star/registry/XRegistryKey.hpp>
#endif

// include our specific addon header to get access to functions and definitions
#include <addon.hxx>

using namespace ::rtl;
using namespace ::osl;
using namespace ::cppu;
using namespace ::com::sun::star::uno;
using namespace ::com::sun::star::lang;
using namespace ::com::sun::star::registry;

//##########################################################################################
//#### EXPORTED ############################################################################
//##########################################################################################


/**
 * Gives the environment this component belongs to.
 */
extern "C" void SAL_CALL component_getImplementationEnvironment(const sal_Char ** ppEnvTypeName,
uno_Environment ** ppEnv)
{
        *ppEnvTypeName = CPPU_CURRENT_LANGUAGE_BINDING_NAME;
}

/**
 * This function creates an implementation section in the registry and another subkey
 *
 * for each supported service.
 * @param pServiceManager   the service manager
 * @param pRegistryKey      the registry key
 */
extern "C" sal_Bool SAL_CALL component_writeInfo(void * pServiceManager, void * pRegistryKey) {
    sal_Bool result = sal_False;

    if (pRegistryKey) {
        try {
            Reference< XRegistryKey > xNewKey(
                reinterpret_cast< XRegistryKey * >( pRegistryKey )->createKey(
                    OUString( RTL_CONSTASCII_USTRINGPARAM("/" IMPLEMENTATION_NAME "/UNO/SERVICES")) ) );

            const Sequence< OUString > & rSNL = Addon_getSupportedServiceNames();
            const OUString * pArray = rSNL.getConstArray();
            for ( sal_Int32 nPos = rSNL.getLength(); nPos--; )
```

```
                xNewKey->createKey( pArray[nPos] );

            return sal_True;
        }
        catch (InvalidRegistryException &) {
            // we should not ignore exceptions
        }
    }
    return result;
}

/**
 * This function is called to get service factories for an implementation.
 *
 * @param pImplName       name of implementation
 * @param pServiceManager a service manager, need for component creation
 * @param pRegistryKey    the registry key for this component, need for persistent data
 * @return a component factory
 */
extern "C" void * SAL_CALL component_getFactory(const sal_Char * pImplName,
        void * pServiceManager, void * pRegistryKey) {
    void * pRet = 0;

    if (rtl_str_compare( pImplName, IMPLEMENTATION_NAME ) == 0) {
        Reference< XSingleServiceFactory > xFactory(createSingleFactory(
            reinterpret_cast< XMultiServiceFactory * >(pServiceManager),
            OUString(RTL_CONSTASCII_USTRINGPARAM(IMPLEMENTATION_NAME)),
            Addon_createInstance,
            Addon_getSupportedServiceNames()));

        if (xFactory.is()) {
            xFactory->acquire();
            pRet = xFactory.get();
        }
    }

    return pRet;
}

//##########################################################################################
//#### Helper functions for the implementation of UNO component interfaces ##################
//##########################################################################################

::rtl::OUString Addon_getImplementationName()
throw (RuntimeException) {
    return ::rtl::OUString ( RTL_CONSTASCII_USTRINGPARAM ( IMPLEMENTATION_NAME ) );
}

sal_Bool SAL_CALL Addon_supportsService( const ::rtl::OUString& ServiceName )
throw (RuntimeException)
{
    return ServiceName.equalsAsciiL( RTL_CONSTASCII_STRINGPARAM ( SERVICE_NAME ) );
}

Sequence< ::rtl::OUString > SAL_CALL Addon_getSupportedServiceNames()
throw (RuntimeException)
{
        Sequence < ::rtl::OUString > aRet(1);
    ::rtl::OUString* pArray = aRet.getArray();
    pArray[0] =  ::rtl::OUString ( RTL_CONSTASCII_USTRINGPARAM ( SERVICE_NAME ) );
    return aRet;
}

Reference< XInterface > SAL_CALL Addon_createInstance( const Reference< XMultiServiceFactory > & rSMgr)
        throw( Exception )
{
        return (cppu::OWeakObject*) new Addon( rSMgr );
}
```

The C++ protocol handler in the example has the implementation name `org.openoffice.Office.addon.example`. It supports the URL protocol schema *org.openoffice.Office.addon.example:* and provides three different URL commands: `Function1`, `Function2` and `Help`.

The protocol handler implements the `IDL:com.sun.star.frame.XDispatch]` interface, so it can return a reference to itself when it is queried for a dispatch object that matches the given URL.

The implementation of the `dispatch()` method below shows how the supported commands are routed inside the protocol handler. Based on the path part of the URL, a simple message box displays which function has been called. The message box is implemented using the UNO toolkit and uses the container windows of the given frame as parent window.

```
#ifndef _Addon_HXX
#include <addon.hxx>
#endif
#ifndef _OSL_DIAGNOSE_H_
#include <osl/diagnose.h>
#endif
#ifndef _RTL_USTRING_HXX_
#include <rtl/ustring.hxx>
#endif
#ifndef _COM_SUN_STAR_LANG_XMULTISERVICEFACTORY_HPP_
#include <com/sun/star/lang/XMultiServiceFactory.hpp>
#endif
#ifndef _COM_SUN_STAR_BEANS_PROPERTYVALUE_HPP_
#include <com/sun/star/beans/PropertyValue.hpp>
#endif
#ifndef _COM_SUN_STAR_FRAME_XFRAME_HPP_
#include <com/sun/star/frame/XFrame.hpp>
#endif
#ifndef _COM_SUN_STAR_FRAME_XCONTROLLER_HPP_
#include <com/sun/star/frame/XController.hpp>
#endif
#ifndef _COM_SUN_STAR_AWT_XTOOLKIT_HPP_
#include <com/sun/star/awt/XToolkit.hpp>
#endif
#ifndef _COM_SUN_STAR_AWT_XWINDOWPEER_HPP_
#include <com/sun/star/awt/XWindowPeer.hpp>
#endif
#ifndef _COM_SUN_STAR_AWT_WINDOWATTRIBUTE_HPP_
#include <com/sun/star/awt/WindowAttribute.hpp>
#endif
#ifndef _COM_SUN_STAR_AWT_XMESSAGEBOX_HPP_
#include <com/sun/star/awt/XMessageBox.hpp>
#endif

using rtl::OUString;
using namespace com::sun::star::uno;
using namespace com::sun::star::frame;
using namespace com::sun::star::awt;
using com::sun::star::lang::XMultiServiceFactory;
using com::sun::star::beans::PropertyValue;
using com::sun::star::util::URL;

// This is the service name an Add-On has to implement
#define SERVICE_NAME "com.sun.star.frame.ProtocolHandler"


/**
  * Show a message box with the UNO based toolkit
  */
static void ShowMessageBox(const Reference< XToolkit >& rToolkit, const Reference< XFrame >& rFrame,
const OUString& aTitle, const OUString& aMsgText)
{
    if ( rFrame.is() && rToolkit.is() )
    {
        // describe window properties.
        WindowDescriptor               aDescriptor;
        aDescriptor.Type              = WindowClass_MODALTOP                              ;
        aDescriptor.WindowServiceName = OUString( RTL_CONSTASCII_USTRINGPARAM( "infobox" ));
        aDescriptor.ParentIndex       = -1                                                ;
        aDescriptor.Parent            = Reference< XWindowPeer >( rFrame->getContainerWindow(),
                                                                  UNO_QUERY )  ;
        aDescriptor.Bounds            = Rectangle(0,0,300,200)                 ;
        aDescriptor.WindowAttributes  = WindowAttribute::BORDER                |
                                        WindowAttribute::MOVEABLE              |
                                        WindowAttribute::CLOSEABLE;

        Reference< XWindowPeer > xPeer = rToolkit->createWindow( aDescriptor );
        if ( xPeer.is() )
        {
            Reference< XMessageBox > xMsgBox( xPeer, UNO_QUERY );
            if ( xMsgBox.is() )
            {
                xMsgBox->setCaptionText( aTitle );
                xMsgBox->setMessageText( aMsgText );
                xMsgBox->execute();
            }
        }
    }
}

//########################################################################################################
//#### Implementation of the ProtocolHandler and Dispatch Interfaces             ###################
//########################################################################################################

// XInitialization
/**
  * Called by the Office framework.
```

```
  * We store the context information
  * given, like the frame we are bound to, into our members.
  */
void SAL_CALL Addon::initialize( const Sequence< Any >& aArguments ) throw ( Exception,
RuntimeException)
{
    Reference < XFrame > xFrame;
    if ( aArguments.getLength() )
    {
        aArguments[0] >>= xFrame;
        mxFrame = xFrame;
    }

    // Create the toolkit to have access to it later
    mxToolkit = Reference< XToolkit >( mxMSF->createInstance(
                                       OUString( RTL_CONSTASCII_USTRINGPARAM(
                                           "com.sun.star.awt.Toolkit" ))), UNO_QUERY );
}

// XDispatchProvider
/**
  * Called by the Office framework.
  * We are ask to query the given URL and return a dispatch object if the URL
  * contains an Add-On command.
  */
Reference< XDispatch > SAL_CALL Addon::queryDispatch( const URL& aURL, const ::rtl::OUString&
sTargetFrameName, sal_Int32 nSearchFlags )
                                  throw( RuntimeException )
{
    Reference < XDispatch > xRet;
    if ( aURL.Protocol.compareToAscii("org.openoffice.Office.addon.example:") == 0 )
    {
        if ( aURL.Path.compareToAscii( "Function1" ) == 0 )
            xRet = this;
        else if ( aURL.Path.compareToAscii( "Function2" ) == 0 )
            xRet = this;
        else if ( aURL.Path.compareToAscii( "Help" ) == 0 )
            xRet = this;
    }

    return xRet;
}

/**
  * Called by the Office framework.
  * We are ask to query the given sequence of URLs and return dispatch objects if the URLs
  * contain Add-On commands.
  */
Sequence < Reference< XDispatch > > SAL_CALL Addon::queryDispatches(
        const Sequence < DispatchDescriptor >& seqDescripts )
        throw( RuntimeException )
{
    sal_Int32 nCount = seqDescripts.getLength();
    Sequence < Reference < XDispatch > > lDispatcher( nCount );

    for( sal_Int32 i=0; i<nCount; ++i )
        lDispatcher[i] = queryDispatch( seqDescripts[i].FeatureURL, seqDescripts[i].FrameName,
seqDescripts[i].SearchFlags );

    return lDispatcher;
}

// XDispatch
/**
  * Called by the Office framework.
  * We are ask to execute the given Add-On command URL.
  */
void SAL_CALL Addon::dispatch( const URL& aURL, const Sequence < PropertyValue >& lArgs ) throw
(RuntimeException)
{
    if ( aURL.Protocol.compareToAscii("org.openoffice.Office.addon.example:") == 0 )
    {
        if ( aURL.Path.compareToAscii( "Function1" ) == 0 )
        {
            ShowMessageBox( mxToolkit, mxFrame,
                            OUString( RTL_CONSTASCII_USTRINGPARAM( "SDK Add-On example" )),
                            OUString( RTL_CONSTASCII_USTRINGPARAM( "Function 1 activated" )) );
        }
        else if ( aURL.Path.compareToAscii( "Function2" ) == 0 )
        {
            ShowMessageBox( mxToolkit, mxFrame,
                            OUString( RTL_CONSTASCII_USTRINGPARAM( "SDK Add-On example" )),
                            OUString( RTL_CONSTASCII_USTRINGPARAM( "Function 2 activated" )) );
        }
        else if ( aURL.Path.compareToAscii( "Help" ) == 0 )
        {
```

```
            // Show info box
            ShowMessageBox( mxToolkit, mxFrame,
                            OUString( RTL_CONSTASCII_USTRINGPARAM( "About SDK Add-On example" )),
                            OUString( RTL_CONSTASCII_USTRINGPARAM( "This is the SDK Add-On example")));
        }
    }
}
/**
  * Called by the Office framework.
  * We are asked to store a status listener for the given URL.
  */
void SAL_CALL Addon::addStatusListener( const Reference< XStatusListener >& xControl, const URL& aURL )
        throw (RuntimeException)
{
}

/**
  * Called by the Office framework.
  * We are asked to remove a status listener for the given URL.
  */
void SAL_CALL Addon::removeStatusListener( const Reference< XStatusListener >& xControl,
        const URL& aURL )
        throw (RuntimeException)
{
}

//#################################################################################################
//#### Implementation of the recommended/mandatory interfaces of a UNO component ##################
//#################################################################################################

// XServiceInfo
::rtl::OUString SAL_CALL Addon::getImplementationName(  )
        throw (RuntimeException)
{
        return Addon_getImplementationName();
}

sal_Bool SAL_CALL Addon::supportsService( const ::rtl::OUString& rServiceName )
        throw (RuntimeException)
{
    return Addon_supportsService( rServiceName );
}

Sequence< ::rtl::OUString > SAL_CALL Addon::getSupportedServiceNames(  )
        throw (RuntimeException)
{
    return Addon_getSupportedServiceNames();
}
```

Configuration

A protocol handler needs configuration entries, which provide the framework with the necessary information to find the handler. The schema of the configuration branch *org.openoffice.Office.ProtocolHandler* defines how to bind handler instances to their URL schemas:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE oor:component-schema SYSTEM "../../../../component-schema.dtd">
<oor:component-schema xmlns:oor="http://openoffice.org/2001/registry"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
oor:name="ProtocolHandler" oor:package="org.openoffice.Office" xml:lang="en-US">
        <templates>
                <group oor:name="Handler">
                        <prop oor:name="Protocols" oor:type="oor:string-list"/>
                </group>
        </templates>
                <component>
                        <set oor:name="HandlerSet" oor:node-type="Handler"/>
                </component>
</oor:component-schema>
```

Each set node entry specifies one protocol handler, using its UNO implementation name. The only property it has is the `Protocols` item. Its type must be `[string-list]` and it contains a list of URL schemas bound to the handler. Wildcards are allowed, otherwise the entire string must match the dispatched URL.

Configuration for vnd.sun.star.framework.ExampleHandler

The following example *ProtocolHandler.xcu* contains the protocol handler configuration for the example's Java protocol handler:

```
<?xml version='1.0' encoding='UTF-8'?>
<oor:component-data oor:name="ProtocolHandler" oor:package="org.openoffice.Office"
xmlns:oor="http://openoffice.org/2001/registry" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <node oor:name="HandlerSet">
                <node oor:name="vnd.sun.star.framework.ExampleHandler" oor:op="replace">
                        <prop oor:name="Protocols">
                                <value>myProtocol_1://* myProtocol_2://*</value>
                        </prop>
                </node>
        </node>
</oor:component-data>
```

The example adds two new URL protocols using wildcards:

```
myProtocol_1://*
myProtocol_2://*
```

Both protocols are bound to the handler implementation `vnd.sun.star.framework.ExampleHandler`. Note that this must be the implementation name of the handler, not the name of the service `com.sun.star.frame.ProtocolHandler` it implements. Because all implementations of the service `com.sun.star.frame.ProtocolHandler` share the same UNO service name, you cannot use this name in the configuration files.

To prevent ambiguous implementation names, the following naming schema for implementation names is frequently used:

```
vnd.<namespace_of_company>.<namespace_of_implementation>.<class_name>
```

e.g. `vnd.sun.star.framework.ExampleHandler`

```
<namespace_of_company>          = sun.star
<namespace_of_implementation>   = framework
<class_name>                    = ExampleHandler
```

An alternative would be the naming convention proposed in *4.4.3 Writing UNO Components - Core Interfaces to Implement - XServiceInfo*:

```
<namespace_of_creator>.comp.<namespace_of_implementation>.<class_name>
```

e.g. `org.openoffice.comp.framework.OProtocolHandler`

All of these conventions are proposals; what matters is:

- use the implementation name in the configuration file, not the general service name "com.sun.star.frame.ProtocolHandler"
- be careful to choose an implementation name that is likely to be unique, and be aware that your handler ceases to function when another developer adds a handler with the same name.

Configuration for org.openoffice.Office.addon.example

The following *ProtocolHandler.xcu* file configures the example's C++ protocol handler with the implementation name *org.openoffice.Office.addon.example* in the configuration branch *org.openoffice.Office.ProtocolHandler* following the same schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<oor:component-data xmlns:oor="http://openoffice.org/2001/registry"
xmlns:xs="http://www.w3.org/2001/XMLSchema" oor:name="ProtocolHandler"
oor:package="org.openoffice.Office">
 <node oor:name="HandlerSet">
  <node oor:name="org.openoffice.Office.addon.example" oor:op="replace">
   <prop oor:name="Protocols" oor:type="oor:string-list">
    <value>org.openoffice.Office.addon.example:*</value>
```

```
   </prop>
  </node>
 </node>
</oor:component-data>
```

The configuration adds one new URL protocol using wildcards:

```
org.openoffice.Office.addon.example:*
```

Based on this URL protocol, the C++ protocol handler can route, for example, a dispatched URL

org.openoffice.Office.addon.example:Function1

to the corresponding target routine. See the implementation of the `dispatch()` method in the `XDispatch` interface of the C++ source fragment above.

Installation

When the office finds a protocol handler implementation for a URL in the configuration files, it asks the global service manager to instantiate that implementation. All components must be registered with the service manager before they can be instantiated. How this is done is described in section *4.9.1 Writing UNO Components - Deployment Options for Components - UNO Package Installation*.

The easiest method to configure and register a new protocol handler in a single step is to use the package installation tool *pkchkg*. A suitable package file for the example protocol handler could contain the following directory structure:

```
ExampleHandler.zip:
    ProtocolHandler.xcu
    windows.plt/
        examplehandler.dll
    solaris_sparc.plt/
        libexamplehandler.so
    linux_x86.plt/
        libexamplehandler.so
```

The *.xcu* file goes into the root of the package, the shared libraries for the various platforms go to their respective *.plt* directories.

The package installation is as simple as changing to the *<OfficePath>/program* directory with a command-line shell and running

```
$ pkgchk /foo/bar/ExampleHandler.zip
```

For an explanation of the package structure and more deployment options please refer to *4.9 Writing UNO Components - Deployment Options for Components*.

4.7.2 Jobs

Overview

A job in OpenOffice.org is a UNO component that can be executed by the job execution environment upon an event. It can read and write its own set of configuration data in the configuration branch *org.openoffice.Office.Jobs*, and it can be activated and deactivated from a certain point in time using special time stamps. It may be started with or without an environment, and it is protected against termination and lifetime issues.

The event that starts a job can be triggered by:

- any code in OpenOffice.org that detects a defined state at runtime and passes an event string to the service `com.sun.star.task.JobExecutor` through its interface method `com.sun.star.task.XJobExecutor:trigger()`. The job executor looks in the configuration of OpenOffice.org if there are any jobs registered for this event and executes them.
- the global document event broadcaster
- the dispatch framework, which provides for a *vnd.star.sun.job:* URL schema to start jobs using a command URL. This URL schema can execute jobs in three different ways: it can issue an *event* for job components that are configured to wait for it, it can call a component by an *alias* that has been given to the component in the configuration or it can execute a job component directly by its *implementation name*.

If you call `trigger()` at the job executor or employ the global event broadcaster, the office needs a valid set of configuration data for every job you want to run. The third approach, to use a *vnd.star.sun.job:* command URL, works with or without prior configuration.

Illustration 37 shows an example job that counts how many times it has been triggered by an event and deactivates itself when it has been executed twice. It uses its own job-specific configuration layer to store the number of times it has been invoked. This value is passed to each newly created job instance as an initialization argument, and can be checked and written back to the configuration. When the counter exceeds two, the job uses the special deactivation feature of the job execution environment. Each job can have a user time stamp and and administrator time stamp to control activation and deactivation. When a job is deactivated, the execution environment updates the user time stamp value, so that subsequent events do not start this job again. It can be enabled by a newer time stamp value in the administration layer.

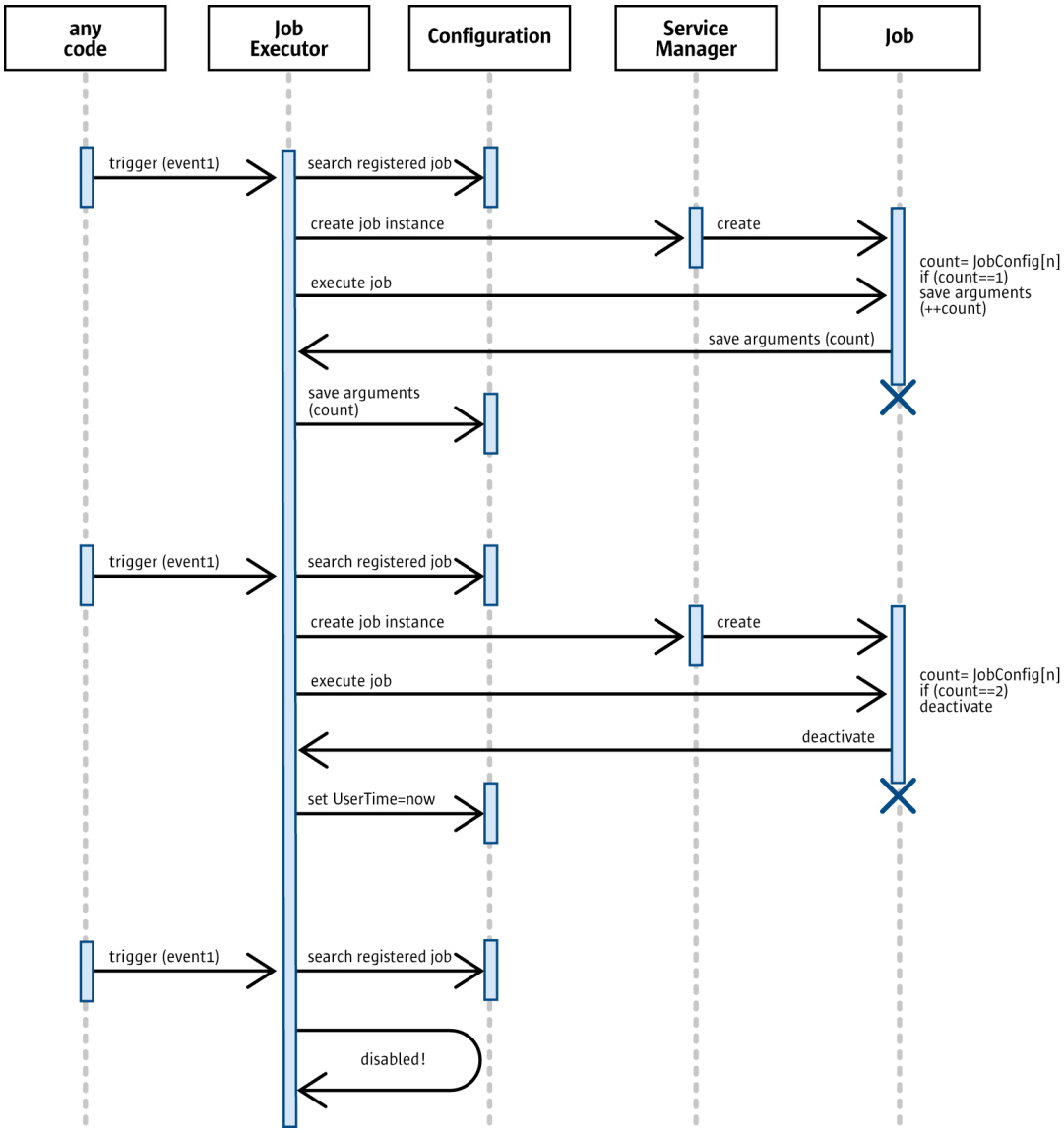


Illustration 38: Flow diagram of an example job

Execution Environment

Jobs are executed in a job execution environment, which handles a number of tasks and problems that can occur when jobs are executed. In particular,

- it initializes the job with all necessary data
- it starts the job using the correct interfaces
- it keeps the job alive by acquiring a UNO reference
- it waits until the job finishes its work, including listening for asynchronous jobs
- it updates the configuration of a job after it has finished
- it informs listeners about the execution

- it protects the job from office termination, or informs it when it is impossible to veto termination

For this purpose, the job execution environment creates special wrapper objects for jobs. This wrapper object implements mechanisms to support lifetime control. The wrapper vetoes termination of the `com.sun.star.frame.Desktop` and the closing of frames that contain document models as long as there are dependent active jobs. It might also register as a `com.sun.star.util.XCloseListener` at a `com.sun.star.frame.Frame` or `com.sun.star.document.OfficeDocument` to handle the close communication on behalf of the job. It also listens for asynchronous job instances, and it is responsible for updates to the configuration data after a job has finished (see *4.7.2 Writing UNO Components - Integrating Components into OpenOffice.org - Jobs - Returning Results*).

A central problem of external components in OpenOffice.org is their lifetime control. Every external component must deal with the possibility that the environment will terminate. It is not efficient to implement lifetime strategies in every job, so the job execution environment takes care of this problem. That way, a job can execute, while difficult situations are handled by the execution environment.

Another advantage of this approach is that it ensures future compatibility. If the mechanism changes in the future, termination is detected and prevented, and it is unnecessary to adapt every existing job implementation.

Implementation

A job must implement the service `com.sun.star.task.Job` if it needs to block the thread in which it is executed or `com.sun.star.task.AsyncJob` if the current state of the office is unimportant for the job. The service that a job implementation supports is detected at runtime. If both are available, the synchronous service `com.sun.star.task.Job` is preferred by the job execution environment.

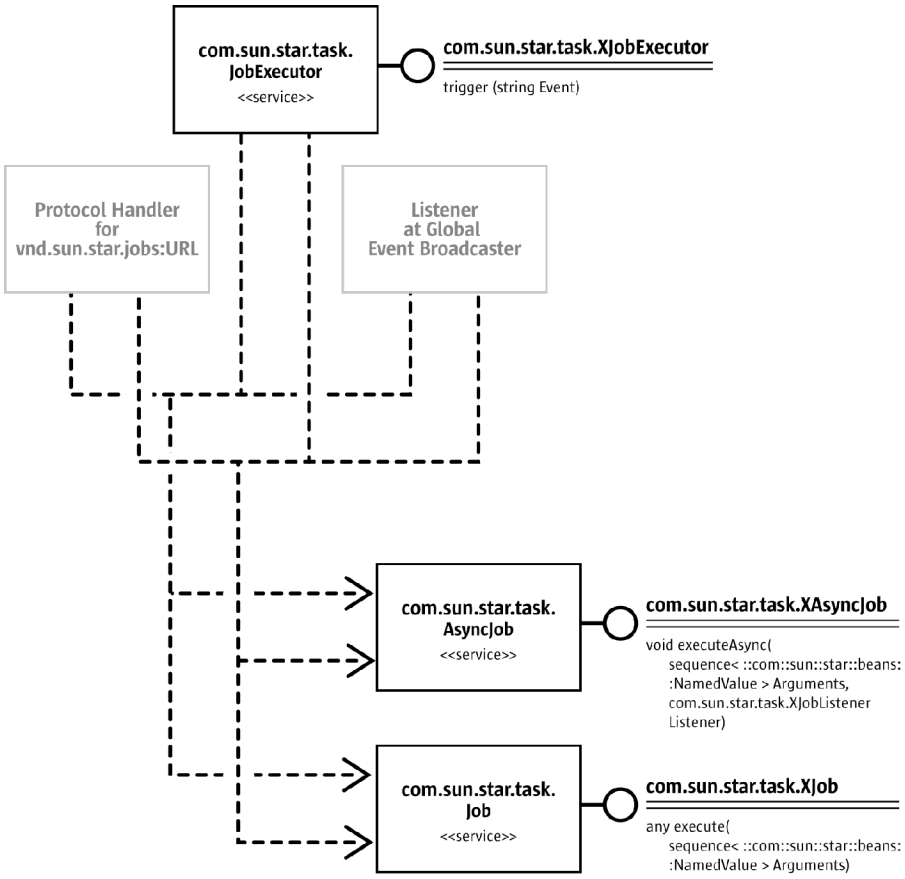


Illustration 39: Job framework

A synchronous job must not make assumptions about the environment, neither that it is the only job that runs currently nor that another object waits for its results. Only the thread context of a synchronous job is blocked until the job finishes its work.

An asynchronous job is not allowed to use threads internally, because OpenOffice.org needs to control thread creation. How asynchronous jobs are executed is an implementation detail of the global job execution environment.

Jobs that need a user interface must proceed with care, so that they do not interfere with the message loop of OpenOffice.org. The following rules apply:

- You cannot display any user interface from a synchronous job, because repaint errors and other threading issues will occur.
- The easiest way to have a user interface for an asynchronous job is to use a non-modal dialog. If you need a modal dialog to get user input, problems can occur. The best way is to use the frame reference that is part of the job environment initialization data, and to get its container window as a parent window. This parent window can be used to create a dialog with the user interface toolkit `com.sun.star.awt.Toolkit`. The C++ protocol handler discussed in *4.7.1 Writing UNO Components - Integrating Components into OpenOffice.org - Protocol Handler - Implementation* shows how a modal message box uses this approach.
- Using a native toolkit or the Java AWT for your GUI can lead to a non-painting OpenOffice.org. To avoid this, the user interface must be non-modal and the implementation must allow the office to abort the job by supporting `com.sun.star.lang.XComponent` or `com.sun.star.util.XCloseable`.

The optional interfaces `com.sun.star.lang.XComponent` or `com.sun.star.util.XCloseable` should be supported so that jobs can be disposed of in a controlled manner. When these interfaces are present, the execution environment can call `dispose()` or `close()` rather than waiting for a job to finish. Otherwise OpenOffice.org must wait until the job is done. Invisible jobs can be especially problematic, because they cannot be recognized as the reason why OpenOffice.org refuses to exit.

Initialization

A job is initialized by a call to its main interface method, which starts the job. For synchronous jobs, the execution environment calls `com.sun.star.task.XJob:execute()`, whereas asynchronous jobs are run through `com.sun.star.task.XAsyncJob:executeAsync()`.

Both methods take one parameter `Arguments`, which is a sequence of `com.sun.star.beans.NamedValue` structs. This sequence describes the job context.

It contains the *environment* where the job is running, which tells if the job was called by the job executor, the dispatch framework or the global event broadcaster service, and possibly provides a frame or a document model for the job to work with.



Section *4.7.1 Writing UNO Components - Integrating Components into OpenOffice.org - Protocol Handler - Implementation* shows how to use a frame to get its associated document model.

The `Arguments` parameter also yields *configuration* data, if the job has been configured in the configuration branch *org.openoffice.Office.Jobs*. This data is separated into basic configuration and additional arguments stored in the configuration. The job configuration is described in section *4.7.2 Writing UNO Components - Integrating Components into OpenOffice.org - Jobs - Configuration*.

Finally, `Arguments` can contain *dynamic parameters* given to the job at runtime. For instance, if a job has been called by the dispatch framework, and the dispatched command URL used parameters, these parameters can be passed on to the job through the execution arguments.

The following table shows the exact specification for the execution `Arguments`:

Elements of the Execution Arguments Sequence		
`Environment`	sequence< com.sun.star.beans.NamedValue >. Contains environment data. The following named values are defined:	
	`EnvType`	string. Determines in which environment a job is executed. Defined Values: `"EXECUTOR"`: job has been executed by a call to `trigger()` at the job executor `"DISPATCH"`: job is dispatched as *vnd.sun.star.job:* URL `"DOCUMENTEVENT"`: job has been executed by the global event broadcaster mechanism
	`Event-Name`	[optional] string. Only exists, if `EnvType` is `"EXECUTOR" or "DOCUMENTEVENT"`. Contains the name of the event for which this job was registered in configuration. During runtime, this information can be used to handle different function sets by the same component implementation.
	`Frame`	[optional] com.sun.star.frame.XFrame. Only exists, if `EnvType` is `"DISPATCH"`. Contains the frame context of this job. Furthermore, the sub list `DynamicData` can contain the optional argument list of the corresponding `com.sun.star.frame.XDispatch:dispatch()` request.
	`Model`	[optional] com.sun.star.frame.XModel. Only exists, if `EnvType` is `"DOCUMENTEVENT"`. Contains the document model that can be used by the job.

Elements of the Execution Arguments Sequence		
Config	[optional] [sequence< com.sun.star.beans.NamedValue >]. Contains the generic set of job configuration properties as described in *4.7.2 Writing UNO Components - Integrating Components into OpenOffice.org - Jobs - Configuration but not* the job specific data set. That is, this sub list only includes the properties Alias and Service, not the property Arguments. The property Arguments is reflected in the element JobConfig (see next element below) Note: this sub list only exists if the job is configured with this data.	
	Alias	string. This property is declared as the name of the corresponding set node in the configuration set *Jobs*. It must be a unique name, which represents the structured information of a job.
	Service	string. Represents the UNO implementation name of the job component.
JobConfig	[optional] [sequence< com.sun.star.beans.NamedValue >] This sub list contains the job-specific set of configuration data as specified in the Arguments property of the job configuration. Its items depend on the job implementation. Note: this sub list only exists if the job is configured with this data.	
DynamicData	[optional] [sequence< com.sun.star.beans.NamedValue >]. Contains optional parameters of the call that started the execution of this job. In particular, it can include the parameters of a com.sun.star.frame.XDispatch:dispatch() request, if Environment-EnvType is "DISPATCH"	

The following example shows how a job can analyze the given arguments and how the environment in which the job is executed can be detected:

```
public synchronized java.lang.Object execute(com.sun.star.beans.NamedValue[] lArgs)
        throws com.sun.star.lang.IllegalArgumentException, com.sun.star.uno.Exception {

    // extract all possible sub list of given argument list
    com.sun.star.beans.NamedValue[] lGenericConfig = null;
    com.sun.star.beans.NamedValue[] lJobConfig     = null;
    com.sun.star.beans.NamedValue[] lEnvironment   = null;
    com.sun.star.beans.NamedValue[] lDispatchArgs  = null;

    int c = lArgs.length;
    for (int i=0; i<c; ++i) {
        if (lArgs[i].Name.equals("Config"))
            lGenericConfig = (com.sun.star.beans.NamedValue[])
                com.sun.star.uno.AnyConverter.toArray(lArgs[i].Value);
        else
            if (lArgs[i].Name.equals("JobConfig"))
                lJobConfig = (com.sun.star.beans.NamedValue[])
                    com.sun.star.uno.AnyConverter.toArray(lArgs[i].Value);
            else
                if (lArgs[i].Name.equals("Environment"))
                    lEnvironment = (com.sun.star.beans.NamedValue[])
                        com.sun.star.uno.AnyConverter.toArray(lArgs[i].Value);
                else
                    if (lArgs[i].Name.equals("DynamicData"))
                        lDispatchArgs = (com.sun.star.beans.NamedValue[])
                            com.sun.star.uno.AnyConverter.toArray(lArgs[i].Value);
                    else
                        // It is not realy an error - because unknown items can be ignored ...
                        throw new com.sun.star.lang.IllegalArgumentException("unknown sub list
detected");
    }

    // Analyze the environment info. This sub list is the only guarenteed one!
    if (lEnvironment==null)
        throw new com.sun.star.lang.IllegalArgumentException("no environment");

    java.lang.String          sEnvType   = null;
    java.lang.String          sEventName = null;
    com.sun.star.frame.XFrame xFrame     = null;
    com.sun.star.frame.XModel xModel     = null;

    c = lEnvironment.length;
    for (int i=0; i<c; ++i) {
        if (lEnvironment[i].Name.equals("EnvType"))
            sEnvType = com.sun.star.uno.AnyConverter.toString(lEnvironment[i].Value);
        else
            if (lEnvironment[i].Name.equals("EventName"))
                sEventName = com.sun.star.uno.AnyConverter.toString(lEnvironment[i].Value);
            else
                if (lEnvironment[i].Name.equals("Frame"))
```

```
                    xFrame = (com.sun.star.frame.XFrame)com.sun.star.uno.AnyConverter.toObject(
                       new com.sun.star.uno.Type(com.sun.star.frame.XFrame.class), lEnvironment[i].
Value);
                else
                    if (lEnvironment[i].Name.equals("Model"))
                        xModel = (com.sun.star.frame.XModel)com.sun.star.uno.AnyConverter.toObject(
                            new com.sun.star.uno.Type(com.sun.star.frame.XModel.class),
                            lEnvironment[i].Value);
    }

    // Further the environment property "EnvType" is required as minimum.
    if (
        (sEnvType==null) ||
        (
         (!sEnvType.equals("EXECUTOR"     )) &&
         (!sEnvType.equals("DISPATCH"     )) &&
         (!sEnvType.equals("DOCUMENTEVENT"))
        )
       )
    {
        throw new com.sun.star.lang.IllegalArgumentException("no valid value for EnvType");
    }

    // Analyze the set of shared config data.
    java.lang.String sAlias = null;
    if (lGenericConfig!=null) {
        c = lGenericConfig.length;
        for (int i=0; i<c; ++i) {
            if (lGenericConfig[i].Name.equals("Alias"))
                sAlias = com.sun.star.uno.AnyConverter.toString(lGenericConfig[i].Value);
        }
    }
}
```

Returning Results

Once a synchronous job has finished its work, it returns its result using the `any` return value of the `com.sun.star.task.XJob:execute()` method. An asynchronous jobs send back the result through the callback method `jobFinished()` to its `com.sun.star.task.XJobListener`. The returned `any` parameter must contain a `sequence< com.sun.star.beans.NamedValue >` with the following elements:

Elements of the Job Return Value	
`Deactivate`	boolean. Asks the job executor to disable a job from further execution. Note that this feature is only available if the next event is triggered by the job executor or the event broadcaster. If it comes, for example, from the dispatch framework using an URL with an `<alias>` argument, the deactivation will be ignored. This value should be used carefully if the `Environment-EnvType` is `"DISPATCH"`, because users will be irritated if clicking a UI element, such as an Add-On menu entry, has no effect.
`SaveArguments`	sequence< com.sun.star.beans.NamedValue >. Must contain a list of job specific data, which are written directly to the `Arguments` list into the job configuration. Note: Merging is not supported. The list must be complete and replaces all values in the configuration. The necessary data can be copied and adjusted from the `JobConfig` element of the execution arguments.
`SendDispatchResult`	com.sun.star.frame.DispatchResultEvent. If a job is designed to be usable in the dispatch framework, this contains a struct, which is send to all interested dispatch result listeners. Tip: This value should be omitted if `Environment-EnvType` is not `"DISPATCH"`.

Configuration

Although jobs that are called through a *vnd.sun.star.jobs:* URL by their implementation name do not require it, a job usually has configuration data. The configuration package *org.openoffice.Office.Jobs* contains all necessary information:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE oor:component-schema SYSTEM "../../../../component-schema.dtd">
<oor:component-schema xmlns:oor="http://openoffice.org/2001/registry"
xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
oor:name="Jobs" oor:package="org.openoffice.Office" xml:lang="en-US">
  <templates>
    <group oor:name="Job">
      <prop oor:name="Service" oor:type="xs:string"/>
      <group oor:name="Arguments" oor:extensible="true"/>
    </group>
    <group oor:name="TimeStamp">
      <prop oor:name="AdminTime" oor:type="xs:string"/>
      <prop oor:name="UserTime" oor:type="xs:string"/>
    </group>
    <group oor:name="Event">
      <set oor:name="JobList" oor:node-type="TimeStamp"/>
    </group>
  </templates>
  <component>
    <set oor:name="Jobs" oor:node-type="Job"/>
    <set oor:name="Events" oor:node-type="Event"/>
  </component>
</oor:component-schema>
```

The `Job` template contains all properties that describe a job component. Instances of this template are located inside the configuration set `Jobs`.

Properties of the Job template	
`Alias`	string. This property is declared as the name of the corresponding set node inside the configuration set `Jobs`. It must be a unique name, which represents the structured information of a job. In the example *.xcu* file below its value is "`SyncJob`". In the job execution arguments this property is passed as `Config - Alias`
`Service`	string. Represents the UNO implementation name of the job component. In the job execution arguments this property is passed as `Config - Service`
`Arguments`	set of `any` entries. This list can be filled with any values and represents the private set of configuration data for this job. In the job execution arguments this property is passed as `JobConfig`

The job property `Alias` was created to provide you with more flexibility for a developing components. You can use the same UNO implementation, but register it with different Aliases. At runtime the job instance will be initialized with its own configuration data and can detect which representation is used.



You cannot use the generic UNO service names `com.sun.star.task.Job` or `com.sun.star.task.AsyncJob` for the `Service` job property, because the job executor cannot identify the correct job implementation. To avoid ambiguities, it is necessary to use the UNO implementation name of the component.

Every job instance can be bound to multiple events. An event indicates a special office state, which can be detected at runtime (for example, `OnFirstVisibleTask`), and which can be triggered by a call to the job executor when the first document window is displayed.

Properties of the Event template	
`EventName`	string. This property is declared as the name of the corresponding set node inside the configuration set `Events`. It must be a unique name, which describes a functional state. In the example *.xcu* file below its value is "`onFirstVisibleTask`". Section *4.7.2 Writing UNO Components - Integrating Components into OpenOffice.org - Jobs - List of Supported Events* summarizes the events currently triggered by the office. In addition, developers can use arbitrary event strings with the *vnd.sun.star.jobs:* URL or in calls to `trigger()` at the `com.sun.star.task.JobExecutor` service.
`JobList`	set of `TimeStamp` entries. This set contains a list of all `Alias` names of jobs that are bound to this event. Every job registration can be combined with time stamp values. Please refer to the description of the template `TimeStamp` below for details

As an optional feature, every job registration that is bound to an event can be enabled or disabled by two time stamp values. In a shared installation of OpenOffice.org, an administrator can use the `AdminTime` value to reactivate jobs for every newly started user office instance; regardless of earlier executions of these jobs. That can be useful, for example, for updating user installations if new functions have been added to the shared installation.

Properties of the TimeStamp template	
`AdminTime`	string. This value must be formatted according to the ISO 8601. It contains the time stamp, which can only be adjusted by an administrator, to reactivate this job.
`UserTime`	string. This value must be formatted according to the ISO 8601. It contains the time, when this job was finished successfully last time upon the configured event.

Using this time stamp feature can sometimes be complicated. For example, assume that there is a job that uses the *pkgchk* mechanism of OpenOffice.org for installation. The job is enabled for a registered event by default, but after the first execution it is disabled. By default, both values (`AdminTime` and `UserTime`) do not exist for a configured event. A *Jobs.xcu* fragment, as part of the package file, must also not contain the `AdminTime` and `UserTime` entries. Because both values are not there, no check can be made and the job is enabled. A job can be deactivated by the global job executor once it has finished its work successfully (depending on the `Deactivate` return value). In that case, the `UserTime` entry is generated and set to the current time. An administrator can set a newer and valid `AdminTime` value in order to reactivate the job again, or the user can remove his `UserTime` entry manually from the configuration file of the user installation.

The following *Jobs.xcu* file shows an example job configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE oor:component-data SYSTEM "../../../../component-update.dtd">
<oor:component-data oor:name="Jobs" oor:package="org.openoffice.Office"
xmlns:oor="http://openoffice.org/2001/registry" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <node oor:name="Jobs">
    <node oor:name="SyncJob" oor:op="replace">
      <prop oor:name="Service">
        <value>com.sun.star.comp.framework.java.services.SyncJob</value>
      </prop>
      <node oor:name="Arguments">
        <prop oor:name=”arg_1” oor:type=”xs:string” oor:op="replace">
          <value>val_1</value>
        </prop>
      </node>
    </node>
  </node>
  <node oor:name="Events">
    <node oor:name="onFirstVisibleTask" oor:op="replace">
      <node oor:name="JobList">
        <node oor:name="SyncJob" oor:op="replace"/>
      </node>
    </node>
  </node>
</oor:component-data>
```

This example job has the following characteristics:

- Its alias name is "SyncJob"
- The UNO implementation name of the component is com.sun.star.comp.framework.java.services.SyncJob.
- The job has its own set of configuration data with one item. It is a string, its name is arg_1 and its value is "val_1".
- The job is bound to the global event onFirstVisibleTask, which is triggered when the first document window of a new OpenOffice.org instance is displayed. The next execution of this job is guaranteed, because there are no time stamp values present.



A job is not executed when it has deactivated itself and is called afterwards by a *vnd.sun.star.jobs:event=...* command URL. This can be confusing to users, especially with add-ons, since it would seem that the customized UI items do not function.

Installation

The easiest way to register an external job component is to use the already mentioned *pkgchk* mechanism of OpenOffice.org, described in section *4.9.1 Writing UNO Components - Deployment Options for Components - UNO Package Installation*. A package file for the example job of this chapter can have the following directory structure:

```
SyncJob.zip:
    Jobs.xcu
    windows.plt/
        SyncJob.jar
```

Using the vnd.sun.star.jobs: URL Schema

This section describes the necessary steps to execute a job by issuing a command URL at the dispatch framework. Based upon the protocol handler mechanism, a specialized URL schema has been implemented in OpenOffice.org. It is registered for the URL schema "vnd.sun.star.jobs:*" which uses the following syntax:

vnd.sun.star.jobs:{[event=<name>]}{,[alias=<name>]}{,[service=<name>]}

Elements of a vnd.sun.star.jobs: URL	
event=<name>	string. Contains an event string, which can also be used as parameter of the interface method com.sun.star.task.XJobExecutor:trigger(). It corresponds to the node name of the set Events in the configuration package *org.openoffice.Office.Jobs*. Using the event parameter of a vnd.sun.star.jobs: URL will start all jobs that are registered for this event in the configuration. Note: Disabled jobs, that is jobs with a user time stamp that is newer than the administrator time stamp, are not triggered by event URLs.
alias=<name>	string. Contains an alias name of a configured job. This name is not used by the job execution API. It is a node name of the set Jobs in the configuration package *org.openoffice.Office.Jobs* Using the alias part of a vnd.sun.star.jobs: URL only starts the requested job.
service=<name>	string. Contains the UNO implementation name of a configured or unconfigured com.sun.star.task.Job or com.sun.star.task.AsyncJob service. It is not necessary that such jobs are registered in the configuration, provided that they work without configuration data or implements necessary configuration on their own.

It is possible to combine elements so as to start several jobs at once with a single URL. For instance, you could dispatch a URL *vnd.sun.star.jobs:event=e1,alias=a1,event=e2,...*. However, URLs that start several jobs at once should be used carefully, since there is no check for double or

concurrent requests. If a service is designed asynchronously, it will be run concurrently with another, synchronous job. If both services work at the same area, there might be race conditions and they must synchronize their work. The generic job execution mechanism does not provide this functionality.

The following configuration file for the configuration package *org.openoffice.Office.Jobs* shows two jobs, which are registered for different events:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE oor:component-data SYSTEM "../../../../component-update.dtd">
<oor:component-data oor:name="Jobs" oor:package="org.openoffice.Office"
xmlns:oor="http://openoffice.org/2001/registry" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <node oor:name="Jobs">
    <node oor:name="Job_1" oor:op="replace">
      <prop oor:name="Service">
        <value>vnd.sun.star.jobs.Job_1</value>
      </prop>
      <node oor:name="Arguments">
        <prop oor:name=”arg_1” oor:type=”xs:string” oor:op="replace">
          <value>val_1</value>
        </prop>
      </node>
    </node>
    <node oor:name="Job_2" oor:op="replace">
      <prop oor:name="Service">
        <value>vnd.sun.star.jobs.Job_2</value>
      </prop>
      <node oor:name="Arguments"/>
    </node>
  </node>
  <node oor:name="Events">
    <node oor:name="onFirstVisibleTask" oor:op="replace">
      <node oor:name="JobList">
        <node oor:name="Job_1" oor:op="replace">
          <prop oor:name="AdminTime">
            <value>01.01.2003/00:00:00</value>
          </prop>
          <prop oor:name="UserTime">
            <value>01.01.2003/00:00:01</value>
          </prop>
        </node>
        <node oor:name="Job_2" oor:op="replace"/>
      </node>
    </node>
  </node>
</oor:component-data>
```

The first job can be described by the following properties:

Properties of “Job_1”	
`alias`	`Job_1`
`UNO implementation name`	`vnd.sun.star.jobs.Job_1`
`activation state`	Disabled for job execution (because its AdminTime is older than its UserTime)
`own configuration`	contains one `string` item `arg1` with the value `"val1"`
`event registration`	job is registered for the event string `"onFirstVisibleTask"`

The second job can be described by these properties:

Properties of “Job_2”	
`alias`	`Job_2`
`UNO implementation name`	`vnd.sun.star.jobs.Job_2`
`activation state`	Enabled for job execution (because it uses default values for `AdminTime` and `UserTime`)
`own configuration`	no own configuration items registered
`event registration`	job is registered for the event string `"onFirstVisibleTask"`

The following demonstrates use cases for all possible *vnd.sun.star.job:* URLs. Not all possible scenarios are shown here. The job dispatch can be used in different ways and the combination of jobs can produce different results:

```
vnd.sun.star.jobs:event=onFirstVisibleTask
```

This URL starts `Job_2` only, `Job_1` is marked `DISABLED`, since its `AdminTime` stamp is older than its `UserTime` stamp.

The job is initialized with environment information through the `Environment` sub list, as shown in section *4.7.2 Writing UNO Components - Integrating Components into OpenOffice.org - Jobs - Initialization*. Optional dispatch arguments are passed in `DynamicData`, and generic configuration data, including the event string, is received in `Config`. However, it is not initialized with configuration data of its own in `JobConfig` because `Job_2` is not configured with such information. On the other hand, `Job_2` may return data after finishing its work, which will be written back to the configuration.

Furthermore, the job instance can expect that the `Frame` property from the `Environment` sub list points to the frame in which the dispatch request is to be executed.

```
vnd.sun.star.jobs:alias=Job_1
```

This starts `Job_1` only. It is initialized with an environment, and optionally initialized with dispatch arguments, generic configuration data, and configuration data of its own. However, the event name is not set here because this job was triggered directly, not using an event name.

```
vnd.sun.star.jobs:service=vnd.sun.star.jobs.Job_3
```

A `vnd.sun.star.jobs.Job_3` is not registered in the job configuration package. However, if this implementation was registered with the global service manager, and if it provided the `com.sun.star.task.XJob` or `com.sun.star.task.XAsyncJob` interfaces, it would be executed by this URL. If both interfaces are present, the synchronous version is preferred.

The given UNO implementation name `vnd.sun.star.jobs.Job_3` is used directly for creation at the UNO service manager. In addition, this job instance is only initialized with an environment and possibly with optional dispatch arguments—there is no configuration data for the job to use.

List of supported Events

Supported events triggered by code	
`onFirstRunInitialization`	Called on startup once after OpenOffice.org is installed. Should be used for post-setup operations.
`onFirstVisibleTask`	Called after a document window has been shown for the first time after launching the application. Note: The quickstarter influences this behavior. With the quickstarter, closing the last document does not close the application. Opening a new document in this situation does not trigger this event.
`onDocumentOpened`	Indicates that a new document was opened. It does not matter if a new or an existing document was opened. Thus it represents the combined `OnNew` and `OnLoad` events of the global event broadcaster.

Supported events triggered by the global event broadcaster	
`OnStartApp`	Application has been started
`OnCloseApp`	Application is going to be closed
`OnNew`	New Document was created
`OnLoad`	Document has been loaded

Supported events triggered by the global event broadcaster	
`OnSaveAs`	Document is going to be saved under a new name
`OnSaveAsDone`	Document was saved under a new name
`OnSave`	Document is going to be saved
`OnSaveDone`	Document was saved
`OnPrepareUnload`	Document is going to be removed
`OnUnload`	Document has been removed
`OnFocus`	Document was activated
`OnUnfocus`	Document was deactivated
`OnPrint`	Document will be printed
`OnModifyChange`	Modified state of the document has changed



Event names are case sensitive.

4.7.3 Add-Ons

A OpenOffice.org add-on is an external UNO component providing one or more functions through the user interface of OpenOffice.org. A typical add-on is available as a UNO package for easier deployment with the *pkgchk* tool. In addition to an ordinary UNO package an add-on package contains configuration files which specify the user interface, registration for a protocol schema and first-time instantiation.

The package installation tool *pkgchk* merges the configuration files with the menu and toolbar items for an add-on directly into the OpenOffice.org configuration files.

Overview

OpenOffice.org supports the integration of add-ons into the following areas of the GUI.

Menu items for add-ons can be added to an **Add-Ons** submenu of the **Tools** menu and a corresponding add-ons popup toolbar icon:

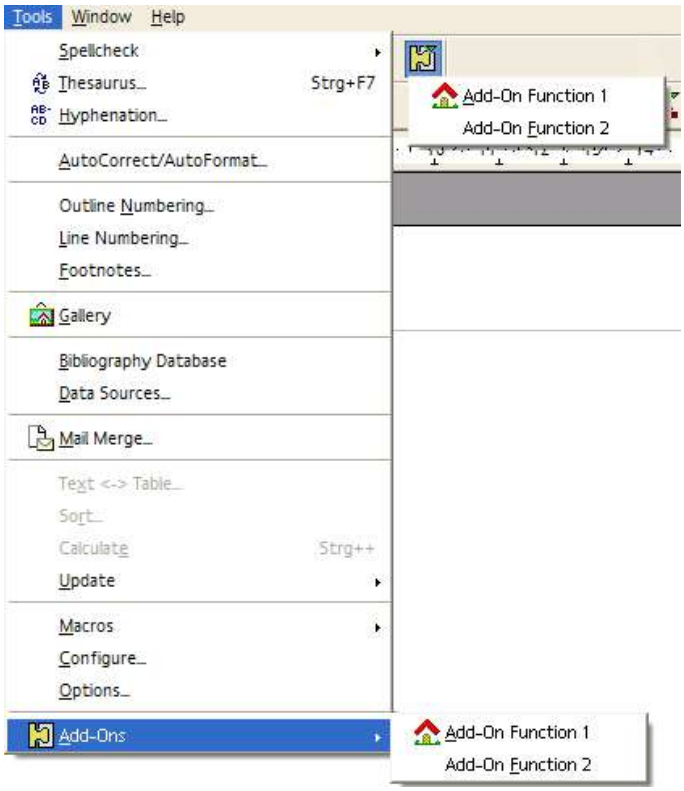


Illustration 40: Add-Ons submenu and toolbar popup

It is also possible to create custom menus in the Menu Bar. You are free to choose your own menu title, and you can create menu items and submenus for your add-on. Custom menus are inserted between the **Tools** and **Window** menus. Separators are supported as well:

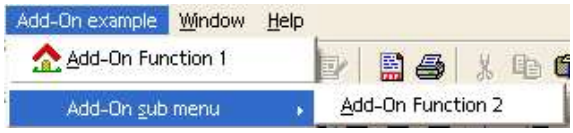


Illustration 41: Custom top-level menu

You can create toolbar icons in the Function Bar, which is usually the topmost toolbar. Below you see two toolbar items, an icon for **Function 1** and a text item for **Function 2**:



Illustration 42: Toolbar icons for Function 1 and Function 2

The **Help** menu offers support for add-ons through help menu items that open the online help of an add-on. They are inserted below the **Help - Registration** item under a separator.

Guidelines

For a smooth integration, a developer should be aware of the following guidelines:

Add-Ons Submenu

- Since the **Tools - Add-Ons** menu is shared by all installed add-ons, an add-on should save space and use a submenu when it has more than two functions. The name of the add-on should be part of the menu item names or the submenu title.
- If your add-on has many menu items, use additional submenus to enhance the overview. Use four to seven entries for a single menu. If you exceed this limit, start creating submenus.

Custom Top-Level Menu

- Only frequently used add-ons or add-ons that offer very important functions in a user environment should use their own top-level menu.
- Use submenus to enhance the overview. Use four to seven entries for a single menu. If you exceed this limit, start creating submenus.
- Use the option to group related items by means of separator items.

Toolbar

- Only important functions should be integrated into the toolbar.
- Use the option to group functions by means of separator items.

Add-On Help menu

Every add-on should provide help to user. This help has to be made available through an entry in the OpenOffice.org **Help** menu. Every add-on should only use a single **Help** menu item.

If the add-on comes with its own dialogs, it should also offer **Help** buttons in the dialogs.

Configuration

The user interface definitions of all add-ons are stored in the special configuration branch *org.openoffice.Office.Addons*.

The schema of the configuration branch *org.openoffice.Office.Addons* specifies how to define a user interface extension.

```
<?xml version='1.0' encoding='UTF-8'?>
<oor:component-schema oor:name="Addons" oor:package="org.openoffice.Office" xml:lang="en-US"
xmlns:oor="http://openoffice.org/2001/registry" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <templates>
        <group oor:name="MenuItem">
            <prop oor:name="URL" oor:type="xs:string"/>
            <prop oor:name="Title" oor:type="xs:string" oor:localized="true"/>
            <prop oor:name="ImageIdentifier" oor:type="xs:string"/>
            <prop oor:name="Target" oor:type="xs:string"/>
            <prop oor:name="Context" oor:type="xs:string"/>
            <set oor:name="Submenu" oor:node-type="MenuItem"/>
        </group>
        <group oor:name="PopupMenu">
            <prop oor:name="Title" oor:type="xs:string" oor:localized="true"/>
            <prop oor:name="Context" oor:type="xs:string"/>
            <set oor:name="Submenu" oor:node-type="MenuItem"/>
        </group>
        <group oor:name="ToolBarItem">
```

```
            <prop oor:name="URL" oor:type="xs:string"/>
            <prop oor:name="Title" oor:type="xs:string" oor:localized="true"/>
            <prop oor:name="ImageIdentifier" oor:type="xs:string"/>
            <prop oor:name="Target" oor:type="xs:string"/>
            <prop oor:name="Context" oor:type="xs:string"/>
        </group>
        <group oor:name="UserDefinedImages">
            <prop oor:name="ImageSmall" oor:type="xs:hexBinary"/>
            <prop oor:name="ImageBig" oor:type="xs:hexBinary"/>
            <prop oor:name="ImageSmallHC" oor:type="xs:hexBinary"/>
            <prop oor:name="ImageBigHC" oor:type="xs:hexBinary"/>
            <prop oor:name="ImageSmallURL" oor:type="xs:string"/>
            <prop oor:name="ImageBigURL" oor:type="xs:string"/>
            <prop oor:name="ImageSmallHCURL" oor:type="xs:string"/>
            <prop oor:name="ImageBigHCURL" oor:type="xs:string"/>
        </group>
        <group oor:name="Images">
            <prop oor:name="URL" oor:type="xs:string"/>
            <node-ref oor:name="UserDefinedImages" oor:node-type="UserDefinedImages"/>
        </group>
        <set oor:name="ToolBarItems" oor:node-type="ToolBarItem"/>
    </templates>
    <component>
        <group oor:name="AddonUI">
            <set oor:name="AddonMenu" oor:node-type="MenuItem"/>
            <set oor:name="Images" oor:node-type="Images"/>
            <set oor:name="OfficeMenuBar" oor:node-type="PopupMenu"/>
            <set oor:name="OfficeToolBar" oor:node-type="ToolBarItems"/>
            <set oor:name="OfficeHelp" oor:node-type="MenuItem"/>
        </group>
    </component>
</oor:component-schema>
```

Menus

As explained in the previous section, OpenOffice.org supports two menu locations where an add-on can be integrated: a top-level menu or the **Tools - Add-Ons** submenu. The configuration branch *org.openoffice.Office.Addons* provides two different nodes for these locations:

Supported sets of org.openoffice.Office.Addons to define an Add-On menu	
`OfficeMenuBar`	A menu defined in this set will be a top-level menu in the OpenOffice.org Menu Bar.
`AddonMenu`	A menu defined in this set will be a pop-up menu which is part of the **Add-Ons** menu item located on the bottom position of the **Tools** menu.

*Submenu in **Tools - Add-Ons***

To integrate add-on menu items into the **Tools – Add-Ons** menu, use the `AddonMenu` set. The `AddonMenu` set consists of nodes of type `MenuItem`. The `MenuItem` node-type is also used for the submenus of a top-level add-on menu.

Properties of template MenuItem	
`oor:name`	string. The name of the configuration node. The name must begin with an ASCII letter character. It must be unique within the `OfficeMenuBar` set. Therefore, it is mandatory to use a schema such as `org.openoffice.<developer>.<product>.<addon name>` or `com.<company>.<product>.<addon name>` to avoid name conflicts. Keep in mind that your configuration file will be merged into the OpenOffice.org configuration branch. You do not know which add-ons, or how many add-ons, are currently installed. The node name of menu items of a submenu must be unique only within their submenu. A configuration set cannot guarantee the order of its entries, so you should use a schema such as string + number, for example “m1”, as the name is used to sort the entries.
`URL`	string. Specifies the command URL that should be dispatched when the user activates the menu entry. It will be ignored if the MenuItem is the title of a a submenu. To define a separator you can use the special command URL "private:separator". A separator ignores all other properties.

Properties of template MenuItem	
`Title`	string. Contains the title of a top-level menu item. This property supports localization: The default string, which is used when OpenOffice.org cannot find a string definition for its current language, uses the `value` element without an attribute. You define a string for a certain language with the `xml:lang` attribute. Assign the language/locale to the attribute, for example `<value xml:lang="en-US">string</value>`.
`ImageIdentifier`	string. Defines an optional image URL that could address an internal OpenOffice.org image or an external user-defined image. The syntax of an internal image URL is: *private:image/<number>* where number specifies the image. External user-defined images are supported using the placeholder variable %origin% representing the folder where the component will be installed by the *pkgchk* tool. The *pkgchk* tool will exchanges %origin% by another placeholder, which is substituted during runtime by OpenOffice.org to the real installation folder. Since OpenOffice.org supports two different configuration folders (*user* and *share*) this mechanism is necessary to determine the installation folder of a component. For example the URL *%origin%/image* will be substituted to something like *vnd.sun.star.expand:$UNO_USER_PACKAGES_CACHE/uno_packages/component.zip.10516 10942/image* . The placeholder *vnd.sun.star.expand:$UNO_USER_PACKAGES_CACHE* will then be substituted during runtime by the real path. As the `ImageIdentifier` property can only hold one URL but OpenOffice.org supports four different images (small/large image and both as high contrast), a naming schema is used to address them. OpenOffice.org adds *_16.bmp* and *_26.bmp* to the provided URL to address the small and large image. *_16h.bmp* and *_26h.bmp* is added to address the high contrast images. If the high contrast images are omitted the normal images are used instead. OpenOffice.org supports bitmaps with 1, 4, 8, 16, 24 bit color depth. Magenta (color value red=0xffff, green=0x0000, blue=0xffff) is used as the transparent color, which means that the background color of the display is used instead of the image pixel color when the image is drawn. For optimal results the size of small images should be 16x16 pixel and for big images 26x26 pixel. Other image sizes are scaled automatically by OpenOffice.org. If no high contrast image is provided, OpenOffice.org uses the normal image for high contrast environments. Images that are not valid will be ignored. This property has a higher priority than the `Images` set when OpenOffice.org searches for images.
`Target`	string. Specifies the target frame for the command URL. Normally an add-on will use one of the predefined target names: _top Returns the top frame of the called frame, which is the first frame where `isTop()` returns true when traversing up the hierarchy. _parent Returns the next frame above in the frame hierarchy. _self Returns the frame itself, same as an empty target frame name. This means you are searching for a frame you already have, but it is legal to do so. _blank Creates a new top-level frame whose parent is the desktop frame.

Properties of template MenuItem	
`Context`	string. A list of service names, separated by a comma, that specifies in which context the add-on menu function should be visible. An empty Context means that the function should visible in all contexts. The OpenOffice.org application modules use the following services names: Writer: com.sun.star.text.TextDocument Spreadsheet: com.sun.star.sheet.SpreadsheetDocument Presentation: com.sun.star.presentation.PresentationDocument Draw: com.sun.star.drawing.DrawingDocument Formula: com.sun.star.formula.FormulaProperties Chart: com.sun.star.chart.ChartDocument Bibliography: com.sun.star.frame.Bibliography The context service name for add-ons is determined by the service name of the model that is bound to the frame, which is associated with UI element (toolbar, menu bar, ...). Thus the service name of the Writer model is `com.sun.star.text.TextDocument`. That means, the context name is bound to the model of an application module. If a developer implements a new desktop component that has a model, it is possible to use its service name as a context for add-on UI items.
`Submenu`	A set of *MenuItem* entries. Optional to define a submenu for the menu entry.

The next examples shows a configuration file specifying a single menu item titled **Add-On Function 1**. The unique node name of the add-on is called *org.openoffice.example.addon.example.function1*.

```
<?xml version='1.0' encoding='UTF-8'?>
<oor:component-data xmlns:oor="http://openoffice.org/2001/registry"
xmlns:xs="http://www.w3.org/2001/XMLSchema" oor:name="Addons" oor:package="org.openoffice.Office">
    <node oor:name="AddonUI">
        <node oor:name="AddonMenu">
            <node oor:name="org.openoffice.Office.addon.example.function1" oor:op="replace">
                <prop oor:name="URL" oor:type="xs:string">
                    <value>org.openoffice.Office.addon.example:Function1</value>
                </prop>
                <prop oor:name="ImageIdentifier" oor:type="xs:string"
                    <value/>
                </prop>
                <prop oor:name="Title" oor:type="xs:string">
                    <value/>
                    <value xml:lang="en-US">Add-On Function 1</value>
                </prop>
                <prop oor:name="Target" oor:type="xs:string">
                    <value>_self</value>
                </prop>
                <prop oor:name="Context" oor:type="xs:string">
                    <value>com.sun.star.text.TextDocument</value>
                </prop>
            </node>
        </node>
    </node>
```

Top-level Menu

If you want to integrate an add-on into the OpenOffice.org Menu Bar, you have to use the `OfficeMenuBar` set. An `OfficeMenuBar` set consists of nodes of type `PopupMenu`.

Properties of template PopupMenu	
`oor:name`	string. The name of the configuration node. The name must begin with an ASCII letter character. It must be unique within the `OfficeMenuBar` set. Therefore, it is mandatory to use a schema such as `org.openoffice.<developer>.<product>.<addon name>` or `com.<company>.<product>.<addon name>` to avoid name conflicts. Please keep in mind that your configuration file will be merged into the OpenOffice.org configuration branch. You do not know what add-ons, or how many add-ons, are currently installed.

Properties of template PopupMenu	
Title	string. Contains the title of a top-level menu item. This property supports localization: The default string, which is used when OpenOffice.org cannot find a string definition for its current language, uses the value element without an attribute. You define a string for a certain language with the xml:lang attribute. Assign the language/locale to the attribute, for example <value xml:lang="en-US">string</value>.
Context	string. A list of service names, separated by a comma, that specifies in which context the add-on menu should be visible. An empty context means that the function should be visible in all contexts. The OpenOffice.org application modules use the following services names: Writer: com.sun.star.text.TextDocument Spreadsheet: com.sun.star.sheet.SpreadsheetDocument Presentation: com.sun.star.presentation.PresentationDocument Draw: com.sun.star.drawing.DrawingDocument Formula: com.sun.star.formula.FormulaProperties Chart: com.sun.star.chart.ChartDocument Bibliography: com.sun.star.frame.Bibliography The context service name for add-ons is determined by the service name of the model that is bound to the frame, which is associated with UI element (toolbar, menu bar, ...). Thus the service name of the Writer model is com.sun.star.text.TextDocument. That means, the context name is bound to the model of an application module. If a developer implements a new desktop component that has a model it is possible to use its service name as a context for add-on UI items.
Submenu	A set of MenuItem entries. Defines the submenu of the top-level menu. It must be defined on a top-level menu otherwise the whole menu will be ignored. For more information how to define a submenu please refer to section *4.7.3 Writing UNO Components - Integrating Components into OpenOffice.org - User Interface Add-Ons - Guidelines* where the MenuItem template is described.

The following example defines a top-level menu titled **Add-On example** with a single menu item titled **Add-On Function 1.** The menu item has a self-defined image used for displaying it next to the menu title.
In the example the nodes are called oor:name="org.openoffice.example.addon" and oor:name="m1".

Do not forget to specify the oor:op="replace" attribute in your self-defined nodes. The replace operation must be used to add a new node to a set or extensible node. Thus the real meaning of the operation is "add or replace". Dynamic properties can only be added once and are then considered mandatory, so during layer merging the replace operation always means "add" for them.
For more details about the configuration and their file formats please read *15 Configuration Management.*

```
<?xml version='1.0' encoding='UTF-8'?>
<oor:component-data xmlns:oor="http://openoffice.org/2001/registry"
xmlns:xs="http://www.w3.org/2001/XMLSchema" oor:name="Addons" oor:package="org.openoffice.Office">
    <node oor:name="AddonUI">
        <node oor:name="OfficeMenuBar">
            <node oor:name="org.openoffice.example.addon" oor:op="replace">
                <prop oor:name="Title" oor:type="xs:string">
                    <value/>
                    <value xml:lang="en-US">Add-On example</value>
                    <value xml:lang="de">Add-On Beispiel</value>
                </prop>
                <prop oor:name="Context" oor:type="xs:string">
                    <value>com.sun.star.text.TextDocument</value>
                </prop>
                <node oor:name="Submenu">
                    <node oor:name="m1" oor:op="replace">
                        <prop oor:name="URL" oor:type="xs:string">
                            <value>org.openoffice.Office.addon.example:Function1</value>
                        </prop>
                        <prop oor:name="Title" oor:type="xs:string">
                            <value/>
```

```
                            <value xml:lang="en-US">Add-On Function 1</value>
                            <value xml:lang="de">Add-On Funktion 1</value>
                        </prop>
                        <prop oor:name="Target" oor:type="xs:string">
                            <value>_self</value>
                        </prop>
                    </node>
                </node>
            </node>
        </node>
    </node>
</oor:component-data>
```

Toolbars

An add-on can also be integrated into the Function Bar of OpenOffice.org. The *org.openoffice.Office.Addons* configuration branch has a set called `OfficeToolBar` where you can add toolbar items for an add-on. The toolbar structure uses an embedded set called `ToolBarItems`, which is used by OpenOffice.org to group toolbar items from different add-ons. OpenOffice.org automatically inserts a separator between different add-ons toolbar items.



The space of the Function Bar is limited, so only the most used/important functions should be added to the `OfficeToolBar` set. Otherwise OpenOffice.org will add scroll-up/down buttons at the end of the Function Bar and the user has to scroll the toolbar to have access to all toolbar buttons.

Properties of template ToolBarItems	
`oor:name`	string. The name of the configuration node. The name must begin with an ASCII letter character. It must be unique within the `OfficeMenuBar` set. Therefore it is mandatory to use a schema such as `org.openoffice.<developer>.<product>.<addon name>` or `com.<company>.<product>.<addon name>` to avoid name conflicts. Please keep in mind that your configuration file will be merged into the OpenOffice.org configuration branch. You do not know what add-ons, or how many add-ons, are currently installed.

The `ToolBarItems` set is a container for the `ToolBarItem` nodes.

Properties of template ToolBarItem	
`oor:name`	string. The name of the configuration node. It must be unique inside your own `ToolBarItems` set. A configuration set cannot guarantee the order of its entries, therefore use a schema such as string + number, for example `"m1"`, as the name is used to sort the entries. Please be aware that the name must begin with an ASCII letter character.
`URL`	string. Specifies the command URL that should be dispatched when the user activates the menu entry. To define a separator you can use the special command URL `"private:separator"`. A separator ignores all other properties.
`Title`	string. Contains the title of a top-level menu item. This property supports localization: The default string, which is used when OpenOffice.org cannot find a string definition for its current language, uses the `value` element without an attribute. You define a string for a certain language with the `xml:lang` attribute. Assign the language/locale to the attribute, for example `<value xml:lang="en-US">string</value>`.

Properties of template ToolBarItem	
`ImageIdentifier`	string. Defines an optional image URL that could address an internal OpenOffice.org image or an external user-defined image. The syntax of an internal image URL is: *private:image/<number>* where number specifies the image. External user-defined images are supported using the placeholder variable %origin%, representing the folder where the component will be installed by the *pkgchk* tool. The *pkgchk* tool exchanges %origin% with another placeholder, which is substituted during runtime by OpenOffice.org to the real installation folder. Since OpenOffice.org supports two different configuration folders (*user* and *share*) this mechanism is necessary to determine the installation folder of a component. For example the URL *%origin%/image* will be substituted with something like *vnd.sun.star.expand:$UNO_USER_PACKAGES_CACHE/uno_packages/component.zip.10516 10942/image* . The placeholder *vnd.sun.star.expand:$UNO_USER_PACKAGES_CACHE* is then substituted during runtime with the real path. Since the `ImageIdentifier` property can only hold one URL but OpenOffice.org supports four different images (small/large image, and both as high contrast), a naming schema is used to address them. OpenOffice.org adds *_16.bmp* and *_26.bmp* to the provided URL to address the small and large image. *_16h.bmp* and *_26h.bmp* is added to address the high contrast images. If the high contrast images are omitted, the normal images are used instead. OpenOffice.org supports bitmaps with 1, 4, 8, 16, 24 bit color depth. Magenta (color value red=0xffff, green=0x0000, blue=0xffff) is used as the transparent color, which means that the background color of the display is used instead of the image pixel color when the image is drawn. For optimal results, the size of small images should be 16x16 pixel, and for big images 26x26 pixel. Other image sizes are scaled automatically by OpenOffice.org. If no high contrast image is provided, OpenOffice.org uses the normal image for high contrast environments. Images that are not valid are ignored. This property has a higher priority than the `Images` set when OpenOffice.org searches for images.
`Target`	string. Specifies the target frame for the command URL. Normally an add-on will use one of the predefined target names: _top Returns the top frame of the called frame, which is the first frame where `isTop()` returns true when traversing up the hierarchy. _parent Returns the next frame above in the frame hierarchy. _self Returns the frame itself, same as an empty target frame name. This means you are searching for a frame you already have, but it is legal to do so. _blank Creates a new top-level frame whose parent is the desktop frame.

Properties of template ToolBarItem	
Context	string. A list of service names, separated by a comma, that specifies in which context the add-on menu should be visible. An empty context means that the function should be visible in all contexts. The OpenOffice.org application modules use the following services names: Writer: com.sun.star.text.TextDocument Spreadsheet: com.sun.star.sheet.SpreadsheetDocument Presentation: com.sun.star.presentation.PresentationDocument Draw: com.sun.star.drawing.DrawingDocument Formula: com.sun.star.formula.FormulaProperties Chart: com.sun.star.chart.ChartDocument Bibliography: com.sun.star.frame.Bibliography The context service name for add-ons is determined by the service name of the model that is bound to the frame, which is associated with an UI element (toolbar, menu bar, ...). Thus the service name of the Writer model is `com.sun.star.text.TextDocument`. That means, the context name is bound to the model of an application module. If you implement a new desktop component that has a model, it is possible to use its service name as a context for add-on UI items.

The following example defines one toolbar button for the function called `org.openoffice.Office.addon.example:Function1`. The toolbar button is only visible when using the OpenOffice.org Writer module.

```
<?xml version='1.0' encoding='UTF-8'?>
<oor:component-data xmlns:oor="http://openoffice.org/2001/registry"
xmlns:xs="http://www.w3.org/2001/XMLSchema" oor:name="Addons" oor:package="org.openoffice.Office">
    <node oor:name="AddonUI">
        <node oor:name="OfficeToolBar">
            <node oor:name="org.openoffice.Office.addon.example" oor:op="replace">
                <node oor:name=”m1”>
                    <prop oor:name="URL" oor:type="xs:string">
                        <value>org.openoffice.Office.addon.example:Function1</value>
                    </prop>
                    <prop oor:name="Title" oor:type="xs:string">
                        <value/>
                        <value xml:lang=”en-US”>Function 1</value>
                        <value xml:lang="de">Funktion 1</value>
                    </prop>
                    <prop oor:name="Target" oor:type="xs:string">
                        <value>_self</value>
                    </prop>
                    <prop oor:name="Context" oor:type="xs:string">
                        <value>com.sun.star.text.TextDocument</value>
                    </prop>
                </node>
            </node>
        </node>
    </node>
</oor:component-data>
```

Images for Toolbars and Menus

OpenOffice.org supports images in menus and toolboxes. In addition to the property ImageIdentifier, the add-ons configuration branch has a fourth set called `Images` that let developers define and use their own images. The image data can be integrated into the configuration either as hex encoded binary data or as references to external bitmap files. The `Images` set binds a *command URL* to user defined images.

Properties of template Images	
`oor:name`	string. The name of the configuration node. It must be unique inside the configuration branch. Therefore it is mandatory to use a schema such as `org.openoffice.<developer>.<add-on name>` or `com.<company>.<product>.<add-on name>` to avoid name conflicts. Please keep in mind that your configuration file will be merged into the OpenOffice.org configuration branch. You do not know how many or which add-ons were installed before by the user. Please be aware that the name must begin with an ASCII letter character.
`URL`	string. Specifies the command URL that should be bound to the defined images. OpenOffice.org searches for images with the command URL that a menu item/toolbox item contains.
`UserDefinedImages`	Group of properties. This optional group provides self-defined images data to OpenOffice.org. There are two different groups of properties to define the image data. One property group provides the image data as ongoing hex values specifying an uncompressed bitmap format stream. The other property group uses URLs to external bitmap files. The names of these properties end with 'URL'. OpenOffice.org supports bitmap streams with 1, 4, 8, 16, 24 bit color depth. Magenta (color value red=0xffff, green=0x0000, blue=0xffff) is used as the transparent color, meaning that the background color of the display will be used instead of the image pixel color when the image is drawn. For best quality, the size of small images should be 16x16 pixel, and for big images 26x26 pixel. Other image sizes will be scaled automatically by OpenOffice.org. If no high contrast image data is provided, OpenOffice.org uses the normal image for high contrast environments. Image data that is not valid will be ignored.

An `Images` node uses a second node called `UserDefinedImages` where the user defined images data are stored.

Properties of template UserDefinedImages	
`ImageSmall`	HexBinary. Used for normal menu/toolbar items, standard size is 16x16 pixel.
`ImageBig`	HexBinary. Only toolbars can use big images. Standard size is 26x26 pixel. The user can activate large buttons with the **Tools – Options – View – Large Buttons** check box.
`ImageSmallHC`	HexBinary. Used for high contrast environments, which means that the background color of a menu or toolbar is below a certain threshold value for the brightness.
`ImageBigHC`	HexBinary. Only toolbars can use big images. Used for high contrast environments, which means that the background color of a toolbar is below a certain threshold value for the brightness.
`ImageSmallURL`	string. An URL to an external image which is used for menu items and normal toolbar buttons. External user-defined images are supported using the placeholder variable %origin%, representing the folder where the component will be installed by the *pkgchk* tool. The *pkgchk* tool exchanges %origin% with another placeholder, which is substituted during runtime by OpenOffice.org to the real installation folder. Since OpenOffice.org supports two different configuration folders (*user* and *share*) this mechanism is necessary to determine the installation folder of a component. For example the URL *%origin%/image* will be substituted with something like *vnd.sun.star.expand:$UNO_USER_PACKAGES_CACHE/uno_packages/component.zip.10516 10942/image* . The placeholder *vnd.sun.star.expand:$UNO_USER_PACKAGES_CACHE* is then substituted during runtime with the real path.
`ImageBigURL`	string. An URL to an external image which is used for big toolbar buttons.

Properties of template UserDefinedImages	
ImageSmallHCURL	string. An URL to an external image which is used for menu items and normal toolbar button in a high contrast environment.
ImageBigHCURL	string. An URL to an external image which is used for big toolbar buttons in a high contrast environment.

The embedded image data have a higher priority when used in conjunction with the URL properties. The embedded and URL properties can be mixed without a problem.

The next example creates two user-defined images for the function `org.openoffice.Office.addon.example:Function1`. The normal image is defined using the embedded image data property *ImageSmall* and has a size of 16x16 pixel and a 4-bit color depth. The other one uses the URL property *ImageSmallHCURL* to reference an external bitmap file for the high contrast image.

```
<?xml version='1.0' encoding='UTF-8'?>
<oor:component-data xmlns:oor="http://openoffice.org/2001/registry"
xmlns:xs="http://www.w3.org/2001/XMLSchema" oor:name="Addons" oor:package="org.openoffice.Office">
    <node oor:name="AddonUI">
        <node oor:name="Images">
            <node oor:name="com.sun.star.comp.framework.addon.image1" oor:op="replace">
                <prop oor:name="URL" oor:type="xs:string">
                    <value>org.openoffice.Office.addon.example:Function1</value>
                </prop>
                <node oor:name=”UserDefinedImages”>
                    <prop oor:name=”ImageSmall”>
                        <value>424df8000000000000076000000280000001000000010000000010004000000000000000
000120b0000120b0000000000000000000000000ff0000ffff0000ff0000ffff0000ff000000ff00ff00ffffff00c0c0c00080808
0000000000000008000008080000080000080800000800000000800080000cccccccccccccccc2c266b181b666c2c5cc66b818b666
5c555566b181b66655555566b818b66655555566b181b6665555a8666bbb6668a55a0a866666668a0a5000a8666668a000a6000a
86668a000a556000a868a000a55556000a8a000a5555556000a000a55555555600000a55555555556000a55555555555560a5555
5550000</value>
                    </prop>
                    <prop oor:name=”ImageSmallHCURL”>
                        <value>%origin%/function1.bmp</value>
                    </prop>
                </node>
            </node>
        </node>
    </node>
</oor:component-data>
```

Help Integration

OpenOffice.org supports the integration of add-ons into its **Help** menu. The add-on help menu items are inserted below the **Registration** menu item, guarded by separators. This guarantees that users have quick access to the add-on help.
The *OfficeHelp* set uses the same *MenuItem* node-type as the *AddonMenu* set, but there are some special treatments of the properties.

Properties of template *MenuItem*	
oor:name	string. The name of the configuration node. It must be unique inside the configuration branch. Therefore it is mandatory to use a schema such as `org.openoffice.<developer>.<add-on name>` or `com.<company>.<product>.<add-on name>` to avoid name conflicts. Please keep in mind that your configuration file will be merged into the OpenOffice.org configuration branch. You do not know how many or which add-ons were installed before by the user. Please be aware that the name must begin with an ASCII letter character.
URL	string. Specifies the help command URL that should be dispatched when the user activates the menu entry. Separators defined by the special command URL "`private:separator`" are supported, but should not be used in the help menu, because every add-on should only use one menu item.

Properties of template *MenuItem*	
`Title`	string. Contains the title of a top-level menu item. This property supports localization: The default string, which is used when OpenOffice.org cannot find a string definition for its current language, uses the `value` element without an attribute. You define a string for a certain language with the `xml:lang` attribute. Assign the language/locale to the attribute, for example `<value xml:lang="en-US">string</value>`.
`ImageIdentifier`	string. Defines an optional image URL that could address an internal OpenOffice.org image or an external user-defined image. The syntax of an internal image URL is: *private:image/<number>* where *number* specifies the image. External user-defined images are supported using the placeholder variable %origin%, representing the folder where the component will be installed by the *pkgchk* tool. The *pkgchk* tool exchanges %origin% with another placeholder, which is substituted during runtime by OpenOffice.org to the real installation folder. Since OpenOffice.org supports two different configuration folders (*user* and *share*), this mechanism is necessary to determine the installation folder of a component. For example the URL *%origin%/image* is substituted with something like *vnd.sun.star.expand:$UNO_USER_PACKAGES_CACHE/uno_packages/component.zip.10516 10942/image* . The placeholder *vnd.sun.star.expand:$UNO_USER_PACKAGES_CACHE* is then substituted during runtime by the real path. Since the `ImageIdentifier` property can only hold one URL but OpenOffice.org supports four different images (small/large image and both as high contrast), a naming schema is used to address them. OpenOffice.org adds *_16.bmp* and *_26.bmp* to the provided URL to address the small and large image. *_16h.bmp* and *_26h.bmp* is added to address the high contrast images. If the high contrast images are omitted, the normal images are used instead. OpenOffice.org supports bitmaps with 1, 4, 8, 16, 24 bit color depth. Magenta (color value red=0xffff, green=0x0000, blue=0xffff) is used as the transparent color, which means that the background color of the display is used instead of the image pixel color when the image is drawn. For optimal results the size of small images should be 16x16 pixel and for big images 26x26 pixel. Other image sizes will be scaled automatically by OpenOffice.org. If no high contrast image is provided, OpenOffice.org uses the normal image for high contrast environments. Images that are not valid are ignored. This property has a higher priority than the `Images` set when OpenOffice.org searches for images.
`Target`	string. Specifies the target frame for the command URL. Normally an add-on will use one of the predefined target names: _top Returns the top frame of the called frame, which is the first frame where `isTop()` returns true when traversing up the hierarchy. _parent Returns the next frame above in the frame hierarchy. _self Returns the frame itself, same as an empty target frame name. This means you are searching for a frame you already have, but it is legal to do so. _blank Creates a new top-level frame whose parent is the desktop frame.

Properties of template *MenuItem*	
`Context`	string. A list of service names, separated by a comma, that specifies in which context the add-on menu should be visible. An empty context means that the function is visible in all contexts. The OpenOffice.org application modules use the following services names: Writer: com.sun.star.text.TextDocument Spreadsheet: com.sun.star.sheet.SpreadsheetDocument Presentation: com.sun.star.presentation.PresentationDocument Draw: com.sun.star.drawing.DrawingDocument Formula: com.sun.star.formula.FormulaProperties Chart: com.sun.star.chart.ChartDocument Bibliography: com.sun.star.frame.Bibliography The context service name for add-ons is determined by the service name of the model that is bound to the frame, which is associated with an UI element (toolbar, menu bar, ...). Thus the service name of the Writer model is `com.sun.star.text.TextDocument`. That means, the context name is bound to the model of an application module. If a developer implements a new desktop component that has a model, it is possible to use its service name as a context for add-on UI items.
`Submenu`	A set of `MenuItem` entries. Not used for `OfficeHelp` `MenuItems`, any definition inside will be ignored.

The following example shows the single help menu item for the add-on example.

```
<?xml version='1.0' encoding='UTF-8'?>
<oor:component-data xmlns:oor="http://openoffice.org/2001/registry"
xmlns:xs="http://www.w3.org/2001/XMLSchema" oor:name="Addons" oor:package="org.openoffice.Office">
    <node oor:name="AddonUI">
        <node oor:name="OfficeHelp">
            <node oor:name="com.sun.star.comp.framework.addon" oor:op="replace">
                <prop oor:name="URL" oor:type="xs:string"
                    <value>org.openoffice.Office.addon.example:Help</value>
                </prop>
                <prop oor:name="ImageIdentifier" oor:type="xs:string">
                    <value/>
                </prop>
                <prop oor:name="Title" oor:type="xs:string">
                    <value xml:lang="de">Über Add-On Beispiel</value>
                    <value xml:lang="en-US">About Add-On Example</value>
                </prop>
                <prop oor:name="Target" oor:type="xs:string">
                    <value>_self</value>
                </prop>
            </node>
        </node>
    </node>
</oor:component-data>
```

Installation

After finishing the implementation of the UNO component and the definition of the user interface part you can create the UNO package. A UNO package can be used by an end-user to install the add-on into OpenOffice.org.

An UNO package is a zip file containing UNO components, type libraries or basic libraries. The *pkgchk* tool unzips all packages found in the package directory into the cache directory, preserving the file structure of the zip file. It also copies all single files recognized in the package directory to the cache directory.

The configuration files that were created for the add-on, such as protocol handler, jobs, and user interface definition must be added to the root of the zip file. The structure of a zip file supporting Windows should resemble the following code:

```
example_addon.zip:
    Addons.xcu
```

```
    ProtocolHandler.xcu
    windows.plt/
        example_addon.dll
```

Before you install the package, make *absolutely* sure there are no running instances of OpenOffice.org. The *pkchk* tool recognizes a running OpenOffice.org in a local installation, but not in a networked installation. Installing into a running office installation might cause inconsistencies and destroy your installation!

The package installation for the example add-on is as simple as changing to the *<OfficePath>/program* directory with a command-line shell and running

```
[<OfficePath>/program] $ pkgchk /foo/bar/example_addon.zip
```

For an explanation of the package structure and more deployment options, please refer to *4.9 Writing UNO Components - Deployment Options for Components.*

4.7.4 Disable Commands

In OpenOffice.org, there may be situations where functions should be disabled to prevent users from changing or destroying documents inadvertently. OpenOffice.org maintains a list of disabled commands that can be maintained by users and developers through the configuration API.

A command request can be created by any object, but in most cases, user interface objects create these requests. Consider, for instance, a toolbox where different functions acting on the office component are presented as buttons. Once a button is clicked, the desired functionality should be executed. If the code assigned to the button is provided with a suitable command URL, the dispatch framework can handle the user action by creating the request and finding a component that can handle it.

The dispatch framework works with the design pattern *chain of responsibility*: everything a component needs to know if it wants to execute a request is the last link in a chain of objects capable of executing requests. If this object gets the request, it checks whether it can handle it or otherwise passes it to the next chain member until the request is executed or the end of the chain is reached. The disable commands implementation is the first chain member and can therefore work as a wall for all disabled commands. They are not be sent to the next chain member, and disappear.

shows how the disable commands feature affects the normal command application flow.

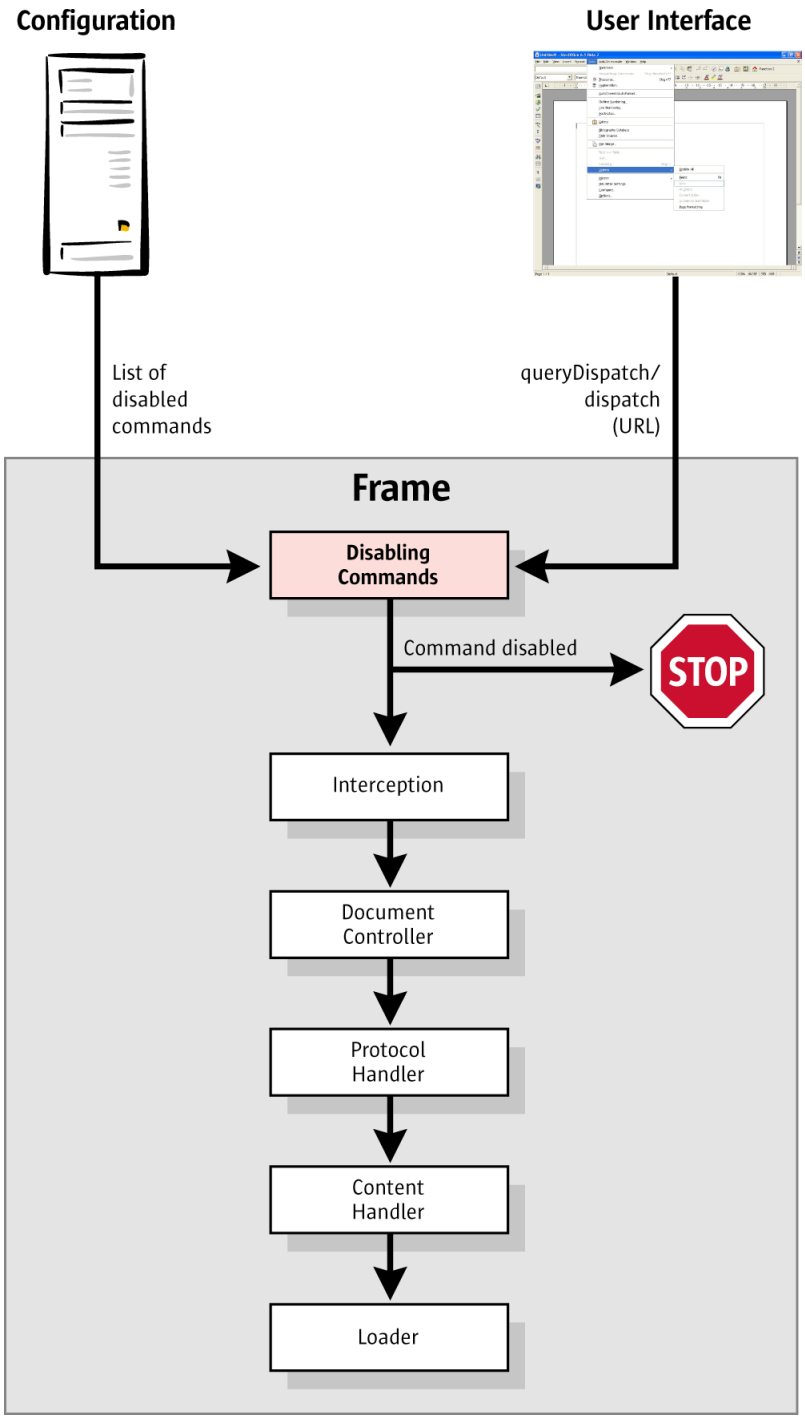


Illustration 43: How the disable commands feature works

Since the disable commands implementation is the first part in the dispatch chain, there is no way to circumvent it. The disabled command must be removed from the list, otherwise it remains disabled.

Configuration

The disable commands feature uses the configuration branch *org.openoffice.Office.Commands* to read which commands should be disabled. The following schema applies:

```
<?xml version='1.0' encoding='UTF-8'?>
<oor:component-schema oor:name="Commands" oor:package="org.openoffice.Office" xml:lang="en-US"
xmlns:oor="http://openoffice.org/2001/registry" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <templates>
        <group oor:name="CommandType">
            <prop oor:name="Command" oor:type="xs:string"/>
        </group>
    </templates>
    <component>
        <group oor:name="Execute">
            <set oor:name="Disabled" oor:node-type="CommandType"/>
        </group>
    </component>
</oor:component-schema>
```

The configuration schema for disabled commands is very simple. The *org.openoffice.Office.Commands* branch has a group called `Execute`. This group has only one set called `Disabled`. The `Disabled` set supports nodes of the type `CommandType`. The following table describes the supported properties of `CommandType`.

Properties of the `CommandType` group	
`oor:component-data`	string. It must be unique inside the `Disabled` set, but has no additional meaning for the implementation of the disable commands feature. Use a consecutive numbering scheme; even numbers are allowed.
`Command`	string. This is the command name with the preceding protocol. That means the command URL *.uno:Open* (which shows the **File – Open** dialog) must be written as `Open`. The valid commands can be found in the document *Index of Command Names* in the Documentation section of the framework project on the OpenOffice.org web page. The OpenOffice.org SDK also includes the latest list of command names.

The example below shows a configuration file that disables the commands for **File – Open**, **Edit – Select All**, **Help – About OpenOffice.org** and **File – Exit**.

```
<?xml version="1.0" encoding="UTF-8" ?>
<oor:component-data oor:name="Commands" oor:package="org.openoffice.Office"
xmlns:oor="http://openoffice.org/2001/registry" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <node oor:name="Execute">
        <node oor:name="Disabled">
            <node oor:name="m1" oor:op="replace">
                <prop oor:name="Command">
                    <value>Open</value>
                </prop>
            </node>
            <node oor:name="m2" oor:op="replace">
                <prop oor:name="Command">
                    <value>SelectAll</value>
                </prop>
            </node>
            <node oor:name="m3" oor:op="replace">
                <prop oor:name="Command">
                    <value>About</value>
                </prop>
            </node>
            <node oor:name="m4" oor:op="replace">
                <prop oor:name="Command">
                    <value>Quit</value>
                </prop>
            </node>
        </node>
    </node>
</oor:component-data>
```

Disabling Commands at Runtime

The following code example first removes all commands that were defined in the user layer of the configuration branch `org.openoffice.Office.Commands` as having a defined starting point. Then it checks if it can get dispatch objects for some pre-defined commands.
Then the example disables these commands and tries to get dispatch objects for them again. At the end, the code removes the disabled commands again, otherwise OpenOffice.org would not be fully useable any longer.

```
import com.sun.star.bridge.XUnoUrlResolver;
import com.sun.star.uno.UnoRuntime;
import com.sun.star.uno.XComponentContext;
import com.sun.star.lang.XMultiComponentFactory;
import com.sun.star.beans.XPropertySet;
import com.sun.star.beans.PropertyValue;
import com.sun.star.lang.XMultiServiceFactory;
import com.sun.star.lang.XSingleServiceFactory;
import com.sun.star.util.XURLTransformer;
import com.sun.star.frame.XDesktop;

import com.sun.star.beans.UnknownPropertyException;

/*
 * Provides example code how to enable/disable
 * commands.
 */
public class DisableCommandsTest extends java.lang.Object {

    /*
     * A list of command names
     */
    final static private String[] aCommandURLTestSet =
    {
        new String( "Open" ),
        new String( "About" ),
        new String( "SelectAll" ),
        new String( "Quit" ),
    };

    private static XComponentContext xRemoteContext = null;
    private static XMultiComponentFactory xRemoteServiceManager = null;
    private static XURLTransformer xTransformer = null;
    private static XMultiServiceFactory xConfigProvider = null;

    /*
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        try {
            // connect
            XComponentContext xLocalContext =
                com.sun.star.comp.helper.Bootstrap.createInitialComponentContext(null);
            XMultiComponentFactory xLocalServiceManager = xLocalContext.getServiceManager();
            Object urlResolver  = xLocalServiceManager.createInstanceWithContext(
                "com.sun.star.bridge.UnoUrlResolver", xLocalContext);
            XUnoUrlResolver xUnoUrlResolver = (XUnoUrlResolver) UnoRuntime.queryInterface(
                XUnoUrlResolver.class, urlResolver );
            Object initialObject = xUnoUrlResolver.resolve(
                "uno:socket,host=localhost,port=8100;urp;StarOffice.ServiceManager");
            XPropertySet xPropertySet = (XPropertySet)UnoRuntime.queryInterface(
                XPropertySet.class, initialObject);
            Object context = xPropertySet.getPropertyValue("DefaultContext");
            xRemoteContext = (XComponentContext)UnoRuntime.queryInterface(
                XComponentContext.class, context);
            xRemoteServiceManager = xRemoteContext.getServiceManager();
            Object transformer = xRemoteServiceManager.createInstanceWithContext(
                "com.sun.star.util.URLTransformer", xRemoteContext);
            xTransformer = (com.sun.star.util.XURLTransformer)UnoRuntime.queryInterface(
                com.sun.star.util.XURLTransformer.class, transformer);

            Object configProvider = xRemoteServiceManager.createInstanceWithContext(
                "com.sun.star.configuration.ConfigurationProvider", xRemoteContext);
            xConfigProvider = (com.sun.star.lang.XMultiServiceFactory)UnoRuntime.queryInterface(
                com.sun.star.lang.XMultiServiceFactory.class, configProvider);

            // First we need a defined starting point. So we have to remove
            // all commands from the disabled set!
            enableCommands();

            // Check if the commands are usable
            testCommands(false);
```

```
        // Disable the commands
        disableCommands();

        // Now the commands should not be usable anymore
        testCommands(true);

        // Remove disable commands to make Office usable again
        enableCommands();
    }
    catch (java.lang.Exception e){
        e.printStackTrace();
    }
    finally {
        System.exit(0);
    }
}

/**
 * Test the commands that we enabled/disabled
 */
private static void testCommands(boolean bDisabledCmds) throws com.sun.star.uno.Exception {
    // We need the desktop to get access to the current frame
    Object desktop = xRemoteServiceManager.createInstanceWithContext(
            "com.sun.star.frame.Desktop", xRemoteContext );
    com.sun.star.frame.XDesktop xDesktop = (com.sun.star.frame.XDesktop)UnoRuntime.queryInterface(
            com.sun.star.frame.XDesktop.class, desktop);
    com.sun.star.frame.XFrame xFrame = xDesktop.getCurrentFrame();
    com.sun.star.frame.XDispatchProvider xDispatchProvider = null;
    if (xFrame != null) {
        // We have a frame. Now we need access to the dispatch provider.
        xDispatchProvider = (com.sun.star.frame.XDispatchProvider)UnoRuntime.queryInterface(
                com.sun.star.frame.XDispatchProvider.class, xFrame );
        if (xDispatchProvider != null) {
            // As we have the dispatch provider we can now check if we get a dispatch
            // object or not.
            for (int n = 0; n < aCommandURLTestSet.length; n++) {
                // Prepare the URL
                com.sun.star.util.URL[] aURL  = new com.sun.star.util.URL[1];
                aURL[0] = new com.sun.star.util.URL();
                com.sun.star.frame.XDispatch xDispatch = null;

                aURL[0].Complete = ".uno:" + aCommandURLTestSet[n];
                xTransformer.parseSmart(aURL, ".uno:");

                // Try to get a dispatch object for our URL
                xDispatch = xDispatchProvider.queryDispatch(aURL[0], "", 0);

                if (xDispatch != null) {
                    if (bDisabledCmds)
                        System.out.println("Something is wrong, I got dispatch object for "
                              + aURL[0].Complete);
                    else
                        System.out.println("Ok, dispatch object for " + aURL[0].Complete);
                }
                else {
                    if (!bDisabledCmds)
                        System.out.println("Something is wrong, I cannot get dispatch object for "
                              + aURL[0].Complete);
                    else
                        System.out.println("Ok, no dispatch object for " + aURL[0].Complete);
                }
                resetURL(aURL[0]);
            }
        }
        else
            System.out.println("Couldn't get XDispatchProvider from Frame!");
    }
    else
        System.out.println("Couldn't get current Frame from Desktop!");
}

/**
 * Ensure that there are no disabled commands in the user layer. The
 * implementation removes all commands from the disabled set!
 */
private static void enableCommands() {
    // Set the root path for our configuration access
    com.sun.star.beans.PropertyValue[] lParams = new com.sun.star.beans.PropertyValue[1];

    lParams[0] = new com.sun.star.beans.PropertyValue();
    lParams[0].Name  = new String("nodepath");
    lParams[0].Value = "/org.openoffice.Office.Commands/Execute/Disabled";

    try {
        // Create configuration update access to have write access to the configuration
        Object xAccess = xConfigProvider.createInstanceWithArguments(
```

```
                "com.sun.star.configuration.ConfigurationUpdateAccess", lParams);

            com.sun.star.container.XNameAccess xNameAccess = (com.sun.star.container.XNameAccess)
                UnoRuntime.queryInterface(com.sun.star.container.XNameAccess.class, xAccess);
            if (xNameAccess != null) {
                // We need the XNameContainer interface to remove the nodes by name
                com.sun.star.container.XNameContainer xNameContainer =
                        (com.sun.star.container.XNameContainer)
                        UnoRuntime.queryInterface(com.sun.star.container.XNameContainer.class, xAccess);

                // Retrieves the names of all Disabled nodes
                String[] aCommandsSeq = xNameAccess.getElementNames();
                for (int n = 0; n < aCommandsSeq.length; n++) {
                    try {
                        // remove the node
                        xNameContainer.removeByName( aCommandsSeq[n]);
                    }
                    catch (com.sun.star.lang.WrappedTargetException e) {
                    }
                    catch (com.sun.star.container.NoSuchElementException e) {
                    }
                }
            }

            // Commit our changes
            com.sun.star.util.XChangesBatch xFlush =
                (com.sun.star.util.XChangesBatch)UnoRuntime.queryInterface(
                com.sun.star.util.XChangesBatch.class, xAccess);
            xFlush.commitChanges();
        }
        catch (com.sun.star.uno.Exception e) {
            System.out.println("Exception detected!");
            System.out.println(e);
        }
    }

    /**
     * Disable all commands defined in the aCommandURLTestSet array
     */
    private static void disableCommands() {
        // Set the root path for our configuration access
        com.sun.star.beans.PropertyValue[] lParams = new com.sun.star.beans.PropertyValue[1];
        lParams[0] = new com.sun.star.beans.PropertyValue();
        lParams[0].Name  = new String("nodepath");
        lParams[0].Value = "/org.openoffice.Office.Commands/Execute/Disabled";

        try {
            // Create configuration update access to have write access to the configuration
            Object xAccess = xConfigProvider.createInstanceWithArguments(
                "com.sun.star.configuration.ConfigurationUpdateAccess", lParams);

            com.sun.star.lang.XSingleServiceFactory xSetElementFactory =
                (com.sun.star.lang.XSingleServiceFactory)UnoRuntime.queryInterface(
                com.sun.star.lang.XSingleServiceFactory.class, xAccess);

            com.sun.star.container.XNameContainer xNameContainer =
                (com.sun.star.container.XNameContainer)UnoRuntime.queryInterface(
                com.sun.star.container.XNameContainer.class, xAccess );

            if (xSetElementFactory != null && xNameContainer != null) {
                Object[] aArgs = new Object[0];

                for (int i = 0; i < aCommandURLTestSet.length; i++) {
                    // Create the nodes with the XSingleServiceFactory of the configuration
                    Object xNewElement = xSetElementFactory.createInstanceWithArguments( aArgs );
                    if (xNewElement != null) {
                        // We have a new node. To set the properties of the node we need
                        // the XPropertySet interface.
                        com.sun.star.beans.XPropertySet xPropertySet =
                            (com.sun.star.beans.XPropertySet)UnoRuntime.queryInterface(
                            com.sun.star.beans.XPropertySet.class,
                            xNewElement );

                        if (xPropertySet != null) {
                            // Create a unique node name.
                            String aCmdNodeName = new String("Command-");
                            aCmdNodeName += i;

                            // Insert the node into the Disabled set
                            xPropertySet.setPropertyValue("Command", aCommandURLTestSet[i]);
                            xNameContainer.insertByName(aCmdNodeName, xNewElement);
                        }
                    }
                }

                // Commit our changes
                com.sun.star.util.XChangesBatch xFlush = (com.sun.star.util.XChangesBatch)
```

```
                    UnoRuntime.queryInterface(com.sun.star.util.XChangesBatch.class, xAccess);
                xFlush.commitChanges();
            }
        }
        catch (com.sun.star.uno.Exception e) {
            System.out.println("Exception detected!");
            System.out.println(e);
        }
    }

    /**
     * reset URL so it can be reused
     *
     * @param aURL
     *          the URL that should be reseted
     */
    private static void resetURL(com.sun.star.util.URL aURL) {
        aURL.Protocol  = "";
        aURL.User      = "";
        aURL.Password  = "";
        aURL.Server    = "";
        aURL.Port      = 0;
        aURL.Path      = "";
        aURL.Name      = "";
        aURL.Arguments = "";
        aURL.Mark      = "";
        aURL.Main      = "";
        aURL.Complete  = "";
    }
}
```

4.8 File Naming Conventions

As a recommendation, UNO component libraries and UNO packages should be named according to the following naming scheme:

`<NAME>[<VERSION>].uno.(so|dll|dylib|jar|zip)`

This recommendation applies to shared libraries and Java archives, as well as UNO packages deployed by *pkgchk* as described in section *4.9.1 Writing UNO Components - Deployment Options for Components - UNO Package Installation.*

This file name convention results in file names such as:

component.uno.so
component1.uno.dll
component0.1.3.uno.dylib
component.uno.jar
component1.5.uno.zip

<NAME> should be a descriptive name, optionally extended by version information as shown below, followed by the characters *.uno* and the necessary file extension.

The term *.uno* is placed next to the platform-specific extension to emphasize that this is a special type of shared library, jar, or zip file.

Usually a shared library or jar has to be registered with UNO to be useful, as its shared library interface only consists of the component operations. Zipped files cannot easily be recognized as UNO packages. In both cases the *.uno* tag informs users that a component or package file is meant for use with UNO.

Since the given naming scheme is only a suggestion, there might be component shared libraries that do not contain the *.uno* addition in their names. Therefore, no tool should build assumptions on whether a shared library name contains *.uno* or not.

<VERSION> is optional and should be in the form:

```
<VERSION>  =  <MAJOR> [.<MINOR> [.<MICRO>]]
<MAJOR>  =  <NUMBER>
```

```
<MINOR>  =  <NUMBER>
<MICRO>  =  <NUMBER>
<NUMBER>  =  0  |  1-9 0-9*
```

Using the version tag in the file name of a shared library or jar is primarily meant for simple components that are not part of a larger UNO package file deployed by *pkgchk*. Such components are usually made up of a single shared library, and different file names for different versions can be useful, for instance in bug reports.

The version of a larger UNO package should be indicated in the package file name, as in *component1.5.uno.zip.*, which results in a corresponding path name in the package cache.

The version of components that are part of the OpenOffice.org installation is already well defined by the version and build number of the installed OpenOffice.org itself.

It is up to the developer how the version scheme is used. You can count versions of a given component shared library using MAJOR alone, or add MINOR and MICRO as needed.



If version is used, it must be placed before the platform-specific extension, never after it. Under Linux and Solaris, there is a convention to add a version number after the *.so*, but that version number has different semantics than the version number used here. In short, those version numbers change whenever the shared library's interface changes, whereas the UNO component interface with the component operations `component_getFactory()` etc. never changes.

The following considerations give an overview of ways that a component can evolve:

A component shared library's interface, as defined by the component operations such as `component_getFactory()` is assumed to be stable.

The UNO services offered by a component can change:

- compatibly: by changing an implementation in the component file but adhering to its specification, or by adding a new UNO service implementation to a component file
- incompatibly: by removing an implementation, or by removing a UNO service from a component
- indirectly compatibly: when one of the UNO services changes compatibility and the component is adapted accordingly. This can happen when a service specification is extended by additional optional interfaces, and the component is altered to support these interfaces.

When an implementation in a component file is changed, for instance when a bug is fixed, such a change will typically be compatible unless clients made themselves dependent on the bug. This can happen when clients considered the bug a feature or worked around the bug in a way that made them dependent on the bug. Therefore developers must be careful to program according to the specification, not the implementation.

Finally, a component shared library can change its dependencies on other shared libraries. Examples of such dependencies are:

C/C++ runtime libraries
: such as *libc.so.6*, *libstdc++.so.3.0.1*, and *libstlport_gcc.so*

UNO runtime libraries
: such as *libcppu.so.3.1.0* and *libcppuhelpergcc3.so.3.1.0*

OpenOffice.org libraries
: such as *libsvx644li.so*

Dependency changes are typically incompatible, as they rely on compatible or incompatible changes of the component's environment.

4.9 Deployment Options for Components

There are a number of opportunities to deploy components to a OpenOffice.org environment. The options available depend on how the new component is to be deployed. If OpenOffice.org is installed in a network mode, the new component could be available to an entire network or to certain users. Another option is to install the new component to individual desktop installations. Third, you may want to use UNO components without any local installation at all. This chapter introduces a simple automatic deployment tool and provides a full understanding of the underlying deployment process, so that you can troubleshoot or deploy manually, if necessary.

4.9.1 UNO Package Installation

OpenOffice.org has a simple concept for adding components to an existing installation. Bringing a UNO component into a OpenOffice.org installation involves the following steps:

- Get a UNO package from a third party vendor or package your component as described below.
- Place the package into a specific *package* directory. By default, the directory for user-specific packages is *<OfficePath>/user/uno_package* and the directory for shared packages in a network installation is *<OfficePath>/share/uno_package*.
- Close all instances of OpenOffice.org, run a command line shell, change to *<OfficePath>/program* and run the tool *pkgchk* from the program directory. The *pkgchk* tool is part of the SDK.

  For a user package, simply run *pkgchk* without options:

  ```
  [<OfficePath>/program] $ pkgchk
  ```

  A shared package is installed using the option `-s` or `--shared`:

  ```
  [<OfficePath>/program] $ pkgchk --shared
  ```

The tool analyzes the packages in the *package* directories, matches them with a cache directory for custom extensions used by OpenOffice.org, registers the components found in the packages and configures them as needed.

As an alternative, the packages can also be specified as command line arguments. In that case, the zip files are automatically copied into the package directory and installed afterwards.

```
[<OfficePath>/program] $ pkgchk /foo/bar/my_package.zip
```

To remove a package from the OpenOffice.org installation, the opposite steps are necessary:

- Remove the package from the packages directory.
- Close all instances of OpenOffice.org and run *pkgchk* or *pkgchk –shared*.

The *pkgchk* mechanism also works for user-defined OpenOffice.org Basic libraries. For details see *11 Basic and Dialogs*.



Be careful not to run the *pkgchk* deployment tool while there are running instances of OpenOffice.org. For ordinary users, this case is recognized by the *pkgchk* process and leads to abortion, but for shared network installations (using option '--shared' or '-s'), this cannot be determined. If any user of a network installation has open processes, data inconsistencies may occur, and OpenOffice.org processes may crash afterwards.

Package Structure

A UNO package is a zip file containing UNO components, type libraries, configuration files or basic libraries. The *pkgchk* tool unzips all packages found in the package directory into the cache directory, preserving the file structure of the zip file. It also copies all single files recognized in the package directory to the cache directory. Subdirectories are normally ignored.

There is often the need for platform dependent files inside a package for the supported UNO platforms. For this purpose, create special platform directories with the extension *.plt* in the package, which are only processed if the platform is present. A package structure for all platforms currently supported by UNO has to look like the following:

```
my_package.zip:
    windows.plt/
        my_comp.dll
    solaris_sparc.plt/
        libmy_comp.so
    linux_x86.plt/
        libmy_comp.so
    linux_powerpc.plt/
        libmy_comp.so
    macosx_powerpc.plt/
        libmy_comp.so
    netbsd_sparc.plt/
        libmy_comp.so
```

A component library might need a third-party library shipped with the package. This library must not be registered if it is not a UNO component. In order to skip registration of shared libraries or jar files, place these libraries in a directory called *skip_registration* within each platform folder. This causes *pkgchk* to copy the files and directories below that directory without registering them.

After the cache directory has been made ready, *pkgchk* traverses the cache directory recursively. Depending on the extension of the files it detects, it carries out the necessary registration steps. Nothing is done for unknown file types.

Shared Libraries

The file extension for shared libraries is *.dll* for Windows and *.so* for Unix. Shared library files are registered and revoked in the registry database *<CacheDir>/services.rdb* and linked into the OpenOffice.org installation through the UNO_SERVICES entry in *uno(.ini|rc)* as shown in the following code. The leading '?' in *uno(.ini|rc)* indicates optional rdb files:

```
UNO_SERVICES=?$UNO_USER_PACKAGES_CACHE/services.rdb \
             ?$UNO_SHARED_PACKAGES_CACHE/services.rdb \
             $SYSBINDIR/applicat.rdb
```

Java Archive Files

Jar files are registered and revoked in the registry database *<CacheDir>/services.rdb* and added to the java classpath of the Java virtual machine used by OpenOffice.org.

Python Components

The UNO deployment tool *pkgchk* now supports registration of Python components (*.py* files). Those files are registered using the `com.sun.star.loader.Python` loader. For details concerning Python-UNO, please refer to *http://udk.openoffice.org/python/python-bridge.html*

Basic Libraries

Basic libraries are recognized by the extension *.xlb*, and they are linked to the basic library container files. Refer to *11 Basic and Dialogs* for additional information.

Type Library Files

The file extension for type libraries is *.rdb* on all platforms. Type libraries in UNO packages are automatically integrated into OpenOffice.org during package installation.

For this purpose, new type library files are merged into the *<CacheDir>/types.rdb* file. In turn, *types.rdb* is linked into the OpenOffice.org installation through the UNO_TYPES entry in the file *uno(.ini|rc)*. The file *uno(.ini|rc)* and its role is described in *4.9.2 Writing UNO Components* -

Deployment Options for Components - Background: UNO Registries. After package installation, *uno (.ini|rc)* contains an entry as shown below. The leading '?' denotes optional type library files in *uno(.ini|rc)*:

```
UNO_TYPES=$SYSBINDIR/applicat.rdb \
          ?$UNO_SHARED_PACKAGES_CACHE/types.rdb \
          ?$UNO_USER_PACKAGES_CACHE/types.rdb
```

Configuration Data Files

Configuration files are recognized by the extension *.xcu* and are mapped into the global OpenOffice.org configuration at runtime.

Configuration Schema Files

Configuration schema files are recognized by the extension *.xcs*. Beware of adding concurring schemata, for example, two *.xcs* files defining the same schema name (`oor:package`, `oor:name`), but different definitions.

Arbitrary Files

Arbitrary files are copied into the UNO packages cache. You can, for instance, deploy an image for add-on menus with the package, or any other file needed by your component. The OpenOffice.org configuration is used to find out in which path this file is located in a particular installation.

When you define a package containing additional files, include an *.xcu* configuration data file, which points to your files. Use a variable `%origin%` as a placeholder for the exact path where the file will be copied by the package installer. When *pkgchk* installs the *.xcu*, it resolves `%origin%` to a file URL in the configuration that points to this path. The URL in the configuration contains a macro that has to be expanded before it is a valid file URL. This can be done using the com.sun.star.util.MacroExpander service. The `%origin%` variable is, for instance, used by the `ImageIdentifier` property of add-on menus and toolbar items, which is described in the *4.7.3 Writing UNO Components - Integrating Components into OpenOffice.org - User Interface Add-Ons - Configuration* section.

Path Settings

The package directories are called *uno-packages* by default. The packages can be in *<OfficePath>/share* for shared installations and another package can be in *<OfficePath>/user* for single users. The cache directories are created automatically within the respective *uno_package* directory. OpenOffice.org must be configured to look for these paths in *uno(.ini|rc)*. When *pkgchk* is launched, it checks *uno(.ini|rc)* for package entries; if they do not exist, the following default values are added:

```
[Bootstrap]
UNO_SHARED_PACKAGES=${$SYSBINDIR/bootstrap.ini::BaseInstallation}/share/uno_packages
UNO_SHARED_PACKAGES_CACHE=$UNO_SHARED_PACKAGES/cache
UNO_USER_PACKAGES=${$SYSBINDIR/bootstrap.ini::UserInstallation}/user/uno_packages
UNO_USER_PACKAGES_CACHE=$UNO_USER_PACKAGES/cache
```

The settings reflect the default values for the *shared* package and cache directory, and the *user* package and cache directory, described previously.

In a network installation, all users start the office from a common directory on a file server. The administrator puts the packages for all users of the network installation into the *<OfficePath>/share/uno_packages* folder of the shared installation. If a user wants to install packages locally so that only the single installation is affected, the packages must be copied to *<OfficePath>/user/uno_packages*.

The *pkgchk* is run differently for a shared and for a user installation. To install shared packages, run *pkgchk* with the `-s (-shared)` option, which causes *pkgchk* to process the shared packages only. If *pkgchk* is run without command-line parameters, the user packages are registered.

Additional Options

The *pkgchk* can be run with the option `-h (--help)` to get a comprehensive overview of all switches.

By default, the tool logs all actions into the *<CacheDir>/log.txt* file. Switch to another log file through the `-l (-log) <file name>` option.

Option `-v (-verbose)` logs to `stdout`, in addition to the log file.

The tool handles errors loosely. It continues after errors, even if a package cannot be inflated or a shared library cannot be registered. The tool logs these errors and proceeds silently. But you can switch on `--strict_error` handling to make the tool stop on every error.

When *pkgchk* is raised giving packages through the command line, then those packages are first copied to the packages directory. Then the packages directory is scanned, balancing the UNO packages cache directory. To avoid the loss of packages of the same name in the packages directory, *pkgchk* aborts if there is an existing file in the packages directory. You have to explicitly raise *pkgchk* with option `-f (--force_overwrite)` to overwrite those files.

If there is inconsistency between the package directory and the cache, renew it from the ground up by repeating the installation, using the option `-r (-renewal)`. This could be necessary when the variables UNO_USER_PACKAGES_CACHE or UNO_SHARED_PACKAGES_CACHE have been modified or after the office installation has been relocated.

4.9.2 Background: UNO Registries

This section explains the necessary steps to deploy new UNO components manually into an installed OpenOffice.org. Background information is provided and the tools required to test deployment are described. The developer and deployer of the component should be familiar with this section. If the recommendations provided are accepted, interoperability of components of different vendors can be achieved easily.

UNO registries store binary data in a tree-like structure. The stored data can be accessed within a registry programmatically through the `com.sun.star.registry.SimpleRegistry` service, however this is generally not necessary. Note that UNO registries have nothing to do with the Windows registry, except that they follow a similar concept for data storage.

UNO-registries mainly store two types of data :

Type-library
: To invoke UNO calls from BASIC or through an interprocess connection, the core UNO bridges need information about the used data types. UNO stores this information into a type library, so that the same data is reusable from any bridge. This is in contrast to the CORBA approach, where code is generated for each data type that needs to be compiled and linked into huge libraries. Every UNOIDL type description is stored as a binary large object (BLOB) that is interpreted by the `com.sun.star.reflection.TypeDescriptionProvider` service.

Information about registered components
: One basic concept of UNO is to create an instance of a component simply by its service name through the `ServiceManager`. The association between the service name and the shared library or *.jar*-file where the necessary compiled code is found is stored into a UNO-registry.
The structure of this data is provided below. Future versions of OpenOffice.org will probably store this information in an XML file that will make it modifiable using a simple text editor.

Both types of data are necessary to run a UNO-C++ process. If the types of data are not present, it could lead to termination of the program. UNO processes in general open their registries during startup and close them when the process terminates. Both types of data are commonly stored in a file with an *.rdb* suffix (rdb=registry database), but this suffix is not mandatory.

UNO Type Library

All type descriptions must be available within the registry under the /UCR main key (UCR = Uno Core Reflection) to be usable in a UNO C++ process . Use the *regview* tool to view the file *<office-path>/install/program/applicat.rdb*. The *regview* tool comes with the OpenOffice.org SDK.

For instance:

```
$ regview applicat.rdb /UCR
```

prints all type descriptions used within the office to `stdout`. To check if a certain type is included within the registry, invoke the following command:

```
$ regview applicat.rdb /UCR/com/sun/star/bridge/XUnoUrlResolver

/UCR/com/sun/star/bridge/XUnoUrlResolver
   Value: Type = RG_VALUETYPE_BINARY
   Size = 461
   Data = minor version: 0
   major version: 1
   type: 'interface'
   name: 'com/sun/star/bridge/XUnoUrlResolver'
   super name: 'com/sun/star/uno/XInterface'
   Doku: ""
   number of fields: 0
   number of methods: 1
   method #0: com/sun/star/uno/XInterface resolve([in] string sUnoUrl)
   raises com/sun/star/connection/NoConnectException,
          com/sun/star/connection/ConnectionSetupException,
          com/sun/star/lang/IllegalArgumentException
   Doku: ""
   number of references: 0
```

The *regview* tool decodes the format of the BLOB containing the type description and presents it in a readable form.

Component Registration

The UNO component provides the data about what services are implemented. In order not to load all available UNO components into memory when starting a UNO process, the data is assembled once during setup and stored into the registry. The process of writing this information into a registry is called *component registration*. The tools used to perform this task are discussed below.

For an installed OpenOffice.org, the *applicat.rdb* contains the component registration information. The data is stored within the /IMPLEMENTATIONS and /SERVICES key. The code below shows a sample SERVICES key for the `com.sun.star.io.Pipe` service.

```
$ regview applicat.rdb /SERVICES/com.sun.star.io.Pipe

/SERVICES/com.sun.star.io.Pipe
  Value: Type = RG_VALUETYPE_STRINGLIST
         Size = 38
         Len  = 1
         Data = 0 = "com.sun.star.comp.io.stm.Pipe"
```

The code above contains one implementation name, but it could contain more than one. In this case, only the first is used. The following entry can be found within the `IMPLEMENTATIONS` section:

```
$ regview applicat.rdb /IMPLEMENTATIONS/com.sun.star.comp.io.stm.Pipe
```

```
/IMPLEMENTATIONS/com.sun.star.comp.io.stm.Pipe
 / UNO
   / ACTIVATOR
     Value: Type = RG_VALUETYPE_STRING
            Size = 34
            Data = "com.sun.star.loader.SharedLibrary"
   / SERVICES
     / com.sun.star.io.Pipe
   / LOCATION
     Value: Type = RG_VALUETYPE_STRING
            Size = 8
            Data = "stm.dll"
```

The implementations section holds three types of data.

1. The loader to be used when the component is requested at runtime (here `com.sun.star.loader.SharedLibrary`).
2. The services supported by this implementation.
3. The URL to the file the loader uses to access the library (the url may be given relative to the OpenOffice.org library directory for native components as it is in this case).

4.9.3 Command Line Registry Tools

There are various tools to create, modify and use registries. This section shows some common use cases. The *regmerge* tool is used to merge multiple registries into a sub-key of an existing or new registry. For instance:

```
$ regmerge new.rdb / test1.rdb test2.rdb
```

merges the contents of *test1.rdb* and *test2.rdb* under the root key / of the registry database *new.rdb* . The names of the keys are preserved, because both registries are merged into the root-key. In case *new.rdb* existed before, the previous contents remain in *new.rdb* unless an identical key names exist in *test1.rdb* and *test2.rdb*. In this case, the content of these keys is overwritten with the ones in *test1.rdb* or *test2.rdb*. So the above command is semantically identical to:

```
$ regmerge new.rdb / test1.rdb
$ regmerge new.rdb / test2.rdb
```

The following command merges the contents of *test1.urd* and *test2.urd* under the key /UCR into the file *myapp_types.rdb*.

```
$ regmerge myapp_types.rdb /UCR test1.urd test2.urd
```

The names of the keys in *test1.urd* and *test2.urd* should only be added to the /UCR key. This is a real life scenario as the files produced by the idl-compiler have a *urd*-suffix. The *regmerge* tool needs to be run before the type library can be used in a program, because UNO expects each type description below the /UCR key.

Component Registration Tool

Components can be registered using the *regcomp* tool. Below, the components necessary to establish an interprocess connection are registered into the *myapp_services.rdb*.

```
$ regcomp -register -r myapp_services.rdb \
                    -c uuresolver.dll             \
                    -c brdgfctr.dll               \
                    -c acceptor.dll               \
                    -c connectr.dll               \
                    -c remotebridge.dll
```

The \ means command line continuation. The option -r gives the registry file where the information is written to. If it does not exist, it is created, otherwise the new data is added. In case there

are older keys, they are overwritten. The registry file (here *myapp_services.rdb*) must NOT be opened by any other process at the same time. The option -c is followed by a single name of a library that is registered. The -c option can be given multiple times. The shared libraries registered in the example above are needed to use the UNO interprocess bridge.

Registering a Java component is currently more complex. It works only in an installed office environment, the <O*fficePath*>*/program* must be the current working directory, the office setup must point to a valid Java installation that can be verified using *jvmsetup* from the <O*fficePath*>*/program*, and Java must be enabled. See **Tools - Options - General - Security**. The office must not run. On Unix, the LD_LIBRARY_PATH environment variable must additionally contain the directories listed by the javaldx tool (which is installed with the office).

Copy the *regcomp* executable into the *<officepath>/program* directory. The *regcomp* tool must then be invoked using the following parameters :

```
$ regcomp -register -r your_registry.rdb \
                    -br applicat.rdb \
                    -l com.sun.star.loader.Java2 \
                    -c file:///i:/StarOffice6.0/program/classes/JavaTestComponent.jar
```

The option -r (registry) tells *regcomp* where to write the registration data and the -br (bootstrap registry) option points *regcomp* to a registry to read common types from. The *regcomp* tool does not know the library that has the Java loader. The -l option gives the service name of the loader to use for the component that must be com.sun.star.loader.Java2. The option can be omitted for C++ components, because *regcomp* defaults to the `com.sun.star.loader.SharedLibrary` loader. The option -c gives the file url to the Java component.

File urls can be given absolute or relative. Absolute file urls must begin with *'file:///'*. All other strings are interpreted as relative file urls. The *'3rdpartYcomp/filterxy.dll'*, *'../../3rdpartycomp/filterxyz.dll'*, and *'filterxyz.dll'* are a few examples. Relative file urls are interpreted relative to all paths given in the PATH variable on Windows and LD_LIBRARY_PATH variable on Unix.

Java components require an *absolute* file URL for historical reasons.



The *regcomp* tool should be used only during the development and testing phase of components. For deploying final components, the *pkgchk* tool should be used instead. See *4.9.1 Writing UNO Components - Deployment Options for Components - UNO Package Installation*.

UNO Type Library Tools

There are several tools that currently access the type library directly. They are encountered when new UNOIDL types are introduced.

- *idlc*, Compiles .idl files into .urd-registry-files.
- *cppumaker*, Generates C++ header for a given UNO type list from a type registry used with the UNO C++ binding.
- *javamaker*, Generates .java files for a given type list from a type registry.
- *rdbmaker*, Creates a new registry by extracting given types (including dependent types) from another registry, and is used for generating minimal, but complete type libraries for components. It is useful when building minimal applications that use UNO components.
- *regcompare*, Compares a type library to a reference type library and checks for compatibility.
- *regmerge*, Merges multiple registries into a certain sub-key of a new or already existing registry.

4.9.4 Manual Component Installation

Manually Merging a Registry and Adding it to uno.ini or soffice.ini

Registry files used by OpenOffice.org are configured within the *uno(.ini|rc)* and *soffice(.ini|rc)* files found in the program directory. After a default OpenOffice.org installation, the files look like this:

```
uno.ini :
[Bootstrap]
UNO_TYPES=$SYSBINDIR/applicat.rdb
UNO_SERVICES=$SYSBINDIR/applicat.rdb

soffice.ini:
[Bootstrap]
Logo=1
UNO_WRITERDB=$SYSUSERCONFIG/user60.rdb
```

The three UNO variables are relevant for UNO components. The UNO_TYPES variable gives a space separated list of type library registries, and the UNO_SERVICES variable gives a space separated list of registries that contain component registration information. These registries are opened read-only. The same registry may appear in UNO_TYPES and UNO_SERVICES variables, for example, the *applicat.rdb*. The UNO_WRITERDB provides one registry that is opened in read-write mode. The $SYSBINDIR points to the directory where the *soffice* executable is located and $SYSUSERCONFIG points to the user's home directory.

OpenOffice.org uses the *applicat.rdb* as a type and component registration information repository. When a programmer or software vendor releases a UNO component, the following files must be provided at a minimum:

- A file containing the code of the new component, for instance a shared library, a jar file, or maybe a python file in the future.
- A registry file containing user defined UNOIDL types, if any.
- (optional) A registry file containing registration information of a pre-registered component. The registry provider should register the component with a relative path to be beneficial in other OpenOffice.org installations.

The latter two can be integrated into a single file.



In fact, a vendor may release more files, such as documentation, the *idl* files of the user defined types, the source code, and configuration files. While every software vendor is encouraged to do this, there are currently no recommendations how to integrate these files into OpenOffice.org. These type of files are ignored in the following paragraphs. These issues will be addressed in next releases of OpenOffice.org.

The recommended method to add a component to OpenOffice.org *manually* is described in the following steps:

1. Copy new shared library components into the *<OfficePath>/program* directory and new Java components into the *<OfficePath>/program/classes* directory.
2. Copy the registry containing the type library into the *<OfficePath>/program* directory, if needed and available.
3. Copy the registry containing the component registration information into the *<OfficePath>/program directory,* if required. Otherwise, register the component with the *regcomp* command line tool coming with the OpenOffice.org SDK into a new registry.
4. Modify the *uno(.ini|rc)* file, and add the type registry to the UNO_TYPES variable and the component registry to the UNO_SERVICES variable. The new *uno(.ini|rc)* might look like this:

```
[Bootstrap]
UNO_TYPES=$SYSBINDIR/applicat.rdb $SYSBINDIR/filterxyz_types.rdb
UNO_SERVICES=$SYSBINDIR/applicat.rdb $SYSBINDIR/filterxyz_services.rdb
```

After these changes are made, every office that is restarted can use the new component. The *uno (.ini|rc)* changes directly affect the whole office network installation. If adding a component only for a single user, pass the modified UNO_TYPES and UNO_SERVICES variables per command line. An example might be:

```
$ soffice "-env:UNO_TYPES=$SYSBINDIR/applicat.rdb $SYSBINDIR/filterxyz_types.rdb"
          "-env:UNO_SERVICES=$SYSBINDIR/applicat.rdb $SYSBINDIR/filter_xyz_services.rdb" ).
```

Alternatives

There are more ways to add a component to the office with their own advantages and disadvantages. Below are some alternatives :

1) When adding many third-party components to your office, the startup performance suffers from having types scattered about many registries. To avoid this, merge all third-party type registries into a single type registry and all third-party service registries into a single service registry using the *regmerge* tool.

   New types and services can be merged into the *applicat.rdb* directly. With this method, the *uno (.ini|rc)* does not have to be modified. Modifying the *applicat.rdb* while there are running office instances is not allowed. This is important in a network installation.
   Once merged, these registries can not be 'unmerge'. To remove a certain type library, a merge with all the other source types will have to be performed, that is, repeat the installation.

2) When separating the OpenOffice.org installation from any third-party additions, the additional registries, shared libraries and .jar files can be stored into a directory other than the *<OfficePath>/program* directory. In this case, use relative filenames for component registrations, for instance *../../office3rdparty/filterxyz.dll*. The only file that needs to be modified is the *uno(.ini|rc)*.

3) Configuring your Application with Bootstrap Parameters

   A flexible approach is to use the UNO bootstrap parameters and the `defaultBootstrap_InitialComponentContext()` function. Arguments, such as registry names are not passed to this function, rather they are given through bootstrap parameters.

4.9.5 Bootstrapping a Service Manager

Bootstrapping a service manager means to create an instance of a service manager that is able to instantiate the UNO objects needed by a user. All UNO applications, that want to use the UnoUrlResolver for connections to the office, have to bootstrap a local service manager in order to create a UnoUrlResolver object. If developers create a new language binding, for instance for a scripting engine, they have to find a way to bootstrap a service manager in the target environment.

There are many methods to bootstrap a UNO C++ application, each requiring one or more registry files to be prepared. Once the registries are prepared, there are different options available to bootstrap your application. A flexible approach is to use UNO bootstrap parameters and the `defaultBootstrap_InitialComponentContext()` function.

```
#include <cppuhelper/bootstrap.hxx>

using namespace com::sun::star::uno;
using namespace com::sun::star::lang;
using namespace rtl;
```

```
using namespace cppu;
int main( )
{
    // create the initial component context
    Reference< XComponentContext > rComponentContext =
        defaultBootstrap_InitialComponentContext();

    // retrieve the service manager from the context
    Reference< XMultiComponentFactory > rServiceManager =
        rComponentContext()->getServiceManager();

    // instantiate a sample service with the service manager.
    Reference< XInterface > rInstance =
        rServiceManger->createInstanceWithContext(
            OUString::createFromAscii("com.sun.star.bridge.UnoUrlResolver" ),
            rComponentContext );

    // continue to connect to the office ....
}
```

No arguments, such as a registry name, are passed to this function. These are given using *bootstrap parameters*. Bootstrap parameters can be passed through a command line, an *.ini* file or using environment variables.

For bootstrapping the UNO component context, the following three variables are relevant:

1) UNO_TYPES
   Gives a space separated list of type library registry files. Each registry must be given as an absolute or relative file url. Note that some special characters within the path require encoding, for example, a space must become a %20. The registries are opened in read-only.

2) UNO_SERVICES
   Gives a space separated list of registry files with component registration information. The registries are opened in read-only. The same registry may appear in UNO_TYPES and UNO_SERVICES variables.

3) UNO_WRITERDB

   Gives one registry file that is opened in read-write mode. Using this variable is optional, because it registers components at runtime and uses them directly.

An absolute file URL must begin with the *file:/// prefix* (on windows, it must look like *file:///c:/mytestregistry.rdb*). To make a file URL relative, the *file:/// prefix* must be omitted. The relative url is interpreted relative to the current working directory.

Within the paths, use special placeholders.

Bootstrap variable	**Meaning**
$SYSUSERHOME	Path of the user's home directory (see osl_getHomeDir())
$SYSBINDIR	Path to the directory of the current executable.
$SYSUSERCONFIG	Path to the directory where the user's configuration data is stored (see osl_getConfigDir())

The advantage of this method is that the executable can be configured after it has been built. The OpenOffice.org bootstraps the service manager with this mechanism.

Consider the following example:

A tool needs to be written that converts documents between different formats. This is achieved by connecting to OpenOffice.org and doing the necessary conversions. The tool is named *docconv*. In the code, the `defaultBootstrap_InitialComponentContext()` function is used as described above to create the component context. Two registries are prepared: *docconv_services.rdb* with the registered components and *applicat.rdb* that contains the types coming with OpenOffice.org. Both files are placed beside the executable. The easiest method to configure the application is to create a

docconv(.ini|rc) ascii file in the same folder as your executable, that contains the following two lines:

```
UNO_TYPES=$SYSBINDIR/applicat.rdb
UNO_SERVICES=$SYSBINDIR/docconv_services.rdb
```

No matter where the application is started form, it will always use the mentioned registries. Note that this also works on different machines when the volume is mapped to different location mount points as $SYSBINDIR is evaluated at runtime.

The second possibility is to set UNO_TYPES and UNO_SERVICES as environment variables, but this method has drawbacks. All UNO applications started with this shell use the same registries.

The third possibility is to pass the variables as command line parameters, for instance

```
docconv -env:UNO_TYPES=$SYSBINDIR/applicat.rdb -env:UNO_SERVICES=$SYSBINDIR/docconv_services.rdb
```

Note that on UNIX shells, you need to quote the $ with a backslash \.

The command line arguments do not need to be passed to the UNO runtime, because it is generally retrieved from some static variables. How this is done depends on the operating system, but it is hidden from the programmer. The *docconv* executable should ignore all command line parameters beginning with '-env:'. The easiest way to do this is to ignore `argc` and `argv[]` and to use the `rtl_getCommandLineArg()` functions defined in *rtl/process.h* header instead which automatically strips the additional parameters.

4) Combine the methods mentioned above. Command line parameters take precedence over *ini* file variables and *.ini* file parameter take precedence over environment variables. That way, it is possible to overwrite the UNO_SERVICES variable on the command line for one invocation of the program only.

4.9.6 Special Service Manager Configurations

The `com.sun.star.container.XSet` interface allows the insertion or removal of `com.sun.star.lang.XSingleServiceFactory` or `com.sun.star.lang.XSingleComponentFactory` implementations into or from the service manager at runtime without making these changes persistent. When the office applications terminate, all the changes are lost. The inserted object must support the `com.sun.star.lang.XServiceInfo` interface. This interface returns the same information as the XServiceInfo interface of the component implementation which is created by the component factory.

With this feature, a running office can be connected, a new factory inserted into the service manager and the new service instantiated without registering it beforehand. This method of hard coding the registered services is not acceptable with OpenOffice.org, because it must be extended after compilation.

Java applications can use a native persistent service manager in their own process using JNI (see *3.4.1 Professional UNO - UNO Language Bindings - Java Language Binding*), or in a remote process. But note, that all services will be instantiated in this remote process.

Dynamically Modifying the Service Manager

Bootstrapping in pure Java is simple, by calling the static runtime method `createInitialComponentContext()` from the Bootstrap class. The following small test program shows how to insert service factories into the service manager at runtime. The sample uses the Java component from

the section *4.5.6 Writing UNO Components - Simple Component in Java - Storing the Service Manager for Further Use*. The complete code can be found with the JavaComp sample component.

The example shows that there is the possibility to control through command line parameter, whether the service is inserted in the local Java service manager or the remote office service manager. If it is inserted into the office service manager, access the service through OpenOffice.org Basic. In both cases, the *component* runs in the local Java process.

If the service is inserted into the office service manager, instantiate the component through OpenOffice.org Basic calling `createUnoService("JavaTestComponentB")`, as long as the Java process is not terminated. Note, to add the new types to the office process by one of the above explained mechanisms, use uno.ini.

```
public static void insertIntoServiceManager(
            XMultiComponentFactory serviceManager, Object singleFactory)
        throws com.sun.star.uno.Exception {
    XSet set = (XSet ) UnoRuntime.queryInterface(XSet.class, serviceManager);
    set.insert(singleFactory);
}

public static void removeFromServiceManager(
            XMultiComponentFactory serviceManager, Object singleFactory)
        throws com.sun.star.uno.Exception {
    XSet set = (XSet) UnoRuntime.queryInterface( XSet.class, serviceManager);
    set.remove(singleFactory);

}

public static void main(String[] args) throws java.lang.Exception {
    if (args.length != 1) {
        System.out.println("usage: RunComponent local|uno-url");
        System.exit(1);
    }
    XComponentContext xLocalComponentContext =
        Bootstrap.createInitialComponentContext(null);

    // initial serviceManager
    XMultiComponentFactory xLocalServiceManager = xLocalComponentContext.getServiceManager();

    XMultiComponentFactory xUsedServiceManager = null;
    XComponentContext xUsedComponentContext = null;
    if (args[0].equals("local")) {
        xUsedServiceManager = xLocalServiceManager;
        xUsedComponentContext = xLocalComponentContext;

        System.out.println("Using local servicemanager");
        // now the local servicemanager is used !
    }
    else {
        // otherwise interpret the string as uno-url
        Object xUrlResolver = xLocalServiceManager.createInstanceWithContext(
            "com.sun.star.bridge.UnoUrlResolver", xLocalComponentContext);
        XUnoUrlResolver urlResolver = (XUnoUrlResolver) UnoRuntime.queryInterface(
            XUnoUrlResolver.class, xUrlResolver);
        Object initialObject = urlResolver.resolve(args[0]);
        xUsedServiceManager = (XmultiComponentFactory) UnoRuntime.queryInterface(
            XMultiComponentFactory.class, initialObject);

        System.out.println("Using remote servicemanager");
        // now the remote servicemanager is used.
    }

    // retrieve the factory for the component implementation
    Object factory = TestServiceProvider.__getServiceFactory(
        "componentsamples.TestComponentB", null, null);

    // insert the factory into the servicemanager
    // from now on, the service can be instantiated !
    insertIntoServiceManager( xUsedServiceManager, factory );

    // Now instantiate one of the services via the servicemanager !
    Object objTest= xUsedServiceManager.createInstanceWithContext(
        "JavaTestComponentB",xUsedComponentContext);

    // query for the service interface
    XSomethingB xs= (XSomethingB) UnoRuntime.queryInterface(
        XSomethingB.class, objTest);

    // and call the test method.
    String s= xs.methodOne("Hello World");
    System.out.println(s);
```

```
        // wait until return is pressed
        System.out.println( "Press return to terminate" );
        while (System.in.read() != 10);

        // remove it again from the servicemanager, otherwise we have
        // a dangling reference ( in case we use the remote service manager )
        removeFromServiceManager( xUsedServiceManager, factory );

        // quit, even when a remote bridge is running
        System.exit(0);
    }
```

Creating a ServiceManager from a Given Registry File

To create a service manager from a given registry, use a single registry that contains the type library and component registration information. Hard code the name of the registry in the program and use the `createRegistryServiceFactory()` function located in the cppuhelper library.

```
#include <cppuhelper/servicefactory.hxx>

using namespace com::sun::star::uno;
using namespace com::sun::star::lang;
using namespace rtl;
using namespace cppu;
int main( )
{
    // create the service manager on the registry test.rdb
    Reference< XMultiServiceFactory > rServiceManager =
        createRegistryServiceFactory( OUString::createFromAscii( "test.rdb" ) );

    // instantiate a sample service with the service manager.
    Reference< XInterface > rInstance =
        rServiceManger->createInstance(
           OUString::createFromAscii("com.sun.star.bridge.UnoUrlResolver" ) );

    // continue to connect to the office ....
}
```



This instantiates the old style service manager without the possibility of offering a component context. In future versions, (642) you will be able to use the new service manager here.

4.10 The UNO Executable

In chapter *3.4.2 Professional UNO - UNO Language Bindings - UNO C++ Binding*, several methods to bootstrap a UNO application were introduced. In this section, the option UNO executable is discussed. With UNO executable, there is no need to write executables anymore, instead only components are developed. Code within executables is *locked up*, it can only run by starting the executable, and it can never be used in another context. Components offer the advantage that they can be used from anywhere. They can be executed from Java or from a remote process.

For these cases, the `com.sun.star.lang.XMain` interface was introduced. It has one method:

```
/* module com.sun.star.lang.XMain */
interface XMain: com::sun::star::uno::XInterface
{
    long run( [in] sequence< string > aArguments );
};
```

Instead of writing an executable, write a component and implement this interface. The component gets the fully initialized service manager during instantiation. The `run()` method then should do what a `main()` function would have done. The UNO executable offers one possible infrastructure for using such components.

Basically, the *uno* tool can do two different things:

1) Instantiate a UNO component which supports the `com.sun.star.lang.XMain` interface and executes the `run()` method.

```
// module com::sun::star::lang
interface XMain: com::sun::star::uno::XInterface
{
    long run( [in] sequence< string > aArguments );
};
```

2) Export a UNO component to another process by accepting on a resource, such as a tcp/ip socket or named pipe, and instantiating it on demand.

In both cases, the *uno* executable creates a UNO component context which is handed to the instantiated component. The registries that should be used are given by command line arguments. The goal of this tool is to minimize the need to write executables and focus on writing components. The advantage for component implementations is that they do not care how the component context is bootstrapped. In the future there may be more ways to bootstrap the component context. While executables will have to be adapted to use the new features, a component supporting `XMain` can be reused.

Standalone Use Case

Simply typing uno gives the following usage screen :

```
uno (-c ComponentImplementationName -l LocationUrl | -s ServiceName)
    [-ro ReadOnlyRegistry1] [-ro ReadOnlyRegistry2] ... [-rw ReadWriteRegistry]
    [-u uno:(socket[,host=HostName][,port=nnn]|pipe[,name=PipeName]);urp;Name
        [--singleaccept] [--singleinstance]]
    [-- Argument1 Argument2 ...]
```

Choosing the implementation to be instantiated

Using the option -s servicename gives the name of the service which shall be instantiated. The uno executable then tries to instantiate a service by this name, using the registries as listed below.

Alternatively, the -l and -c options can be used. The -l gives an url to the location of the shared library or *.jar* file, and -c the name of the desired service implementation inside the component. Remember that a component may contain more than one implementation.

Choosing the registries for the component context (optional)

With the option -ro, give a file url to a registry file containing component's registration information and/or type libraries. The -ro option can be given multiple times. The -rw option can only be given once and must be the name of a registry with read/write access. It will be used when the instantiated component tries to register components at runtime. This option is rarely needed.

Note that the uno tool ignores bootstrap variables, such as UNO_TYPES and UNO_SERVICES.

The UNO URL (optional)

Giving a UNO URL causes the uno tool to start in server mode, then it accepts on the connection part of the UNO URL. In case another process connects to the resource (tcp/ip socket or named pipe), it establishes a UNO interprocess bridge on top of the connection (see also *3.3.1 Professional UNO - UNO Concepts - UNO Interprocess Connections*). Note that *urp* should always be used as protocol. An instance of the component is instantiated when the client requests a named object using the name, which was given in the last part of the UNO URL.

Option `--singleaccept`

Only meaningful when a UNO URL is given. It tells the *uno* executable to accept only one connection, thus blocking any further connection attempts.

Option `--singleinstance`

Only meaningful when a UNO URL is given. It tells the uno executable to always return the same (first) instance of the component, thus multiple processes communicate to the same instance of the implementation. If the option is not given, every `getInstance()` call at the `com.sun.star.bridge.XBridge` interface instantiates a new object.

Option `--` (double dash)

Everything following `--` is interpreted as an option for the component itself. The arguments are passed to the component through the `initialize()` call of `com.sun.star.lang.XInitialization` interface.



The uno executable currently does not support the bootstrap variable concept as introduced by *3.4.2 Professional UNO - UNO Language Bindings - UNO C++ Binding*. The uno registries must be given explicitly given by command line.

The following example shows how to implement a Java component suitable for the uno executable.

```
import com.sun.star.uno.XComponentContext;
import com.sun.star.comp.loader.FactoryHelper;
import com.sun.star.lang.XSingleServiceFactory;
import com.sun.star.lang.XMultiServiceFactory;
import com.sun.star.registry.XRegistryKey;

public class UnoExeMain implements com.sun.star.lang.XMain
{
    final static String __serviceName = "MyMain";
    XComponentContext _ctx;

    public UnoExeMain( XComponentContext ctx )
    {
        // in case we would need the component context !
        _ctx = ctx;
    }

    public int run( /*IN*/String[] aArguments )
    {
        System.out.println( "Hello world !" );
        return 0;
    }

    public static XSingleServiceFactory __getServiceFactory(
        String implName, XMultiServiceFactory multiFactory, XRegistryKey regKey)
    {
        XSingleServiceFactory xSingleServiceFactory = null;

        if (implName.equals(UnoExeMain.class.getName()))
        {
            xSingleServiceFactory =
                FactoryHelper.getServiceFactory(
                    UnoExeMain.class, UnoExeMain.__serviceName, multiFactory, regKey);
        }
        return xSingleServiceFactory;
    }

    public static boolean __writeRegistryServiceInfo(XRegistryKey regKey)
    {
        boolean b = FactoryHelper.writeRegistryServiceInfo(
            UnoExeMain.class.getName(),
            UnoExeMain.__serviceName, regKey);
        return b;
    }
}
```

The class itself inherits from `com.sun.star.lang.XMain`. It implements a constructor with the `com.sun.star.uno.XComponentContext` interface and stores the component context for future use. Within its `run()` method, it prints 'Hello World'. The last two mandatory functions are responsible for instantiating the component and writing component information into a registry. Refer to *4.5.6 Writing UNO Components - Simple Component in Java - Storing the Service Manager for Further Use* for further information.

The code needs to be compiled and put into a *jar* file with an appropriate manifest file:

```
RegistrationClassName: UnoExeMain
```

These commands create the jar:

```
javac UnoExeMain
jar -cvfm UnoExeMain.jar Manifest UnoExeMain.class
```

To be able to use it, register it with the following command line into a separate registry file (here *test.rdb*). The *<OfficePath>/program* directory needs to be the current directory, and the *regcomp* and *uno* tools must have been copied into this directory.

```
regcomp -register \
        -br applicat.rdb \
        -r test.rdb \
        -c file:///c:/devmanual/Develop/samples/unoexe/UnoExeMain.jar \
        -l com.sun.star.loader.Java2
```

The \ means command line continuation.

The component can now be run:

```
uno -s MyMain -ro applicat.rdb -ro test.rdb
```

This command should give the output "hello world !"

Server Use Case

This use case enables the export of any arbitrary UNO component as a remote server. As an example, the `com.sun.star.io.Pipe` service is used which is already implemented by a component coming with the office. It exports an `com.sun.star.io.XOutputStream` and a `com.sun.star.io.XInputStream` interface. The data is written through the output stream into the pipe and the same data from the input stream is read again. To export this component as a remote server, switch to the *<OfficePath>/program* directory and issue the following command line.

```
i:\o641l\program>uno -s com.sun.star.io.Pipe -ro applicat.rdb -u uno:socket,host=0,port=2002;urp;test

> accepting socket,host=0,port=2002...
```

Now a client program can connect to the server. A client may look like the following:

```
import com.sun.star.lang.XServiceInfo;
import com.sun.star.uno.XComponentContext;
import com.sun.star.bridge.XUnoUrlResolver;
import com.sun.star.io.XOutputStream;
import com.sun.star.io.XInputStream;
import com.sun.star.uno.UnoRuntime;

// Note: This example does not do anything meaningful, it shall just show,
//       how to import an arbitrary UNO object from a remote process.
class UnoExeClient {
    public static void main(String [] args) throws java.lang.Exception {
        if (args.length != 1) {
            System.out.println("Usage : java UnoExeClient uno-url");
            System.out.println("   The imported object must support the com.sun.star.io.Pipe service");
            return;
        }

        XComponentContext ctx =
            com.sun.star.comp.helper.Bootstrap.createInitialComponentContext(null);

        // get the UnoUrlResolver service
        Object o = ctx.getServiceManager().createInstanceWithContext(
            "com.sun.star.bridge.UnoUrlResolver" , ctx);
        XUnoUrlResolver resolver = (XUnoUrlResolver) UnoRuntime.queryInterface(
            XUnoUrlResolver.class, o);

        // connect to the remote server and retrieve the appropriate object
        o = resolver.resolve(args[0]);

        // Check if we got what we expected
        // Note: This is not really necessary, you can also use the try and error approach
        XServiceInfo serviceInfo = (XServiceInfo) UnoRuntime.queryInterface(XServiceInfo.class,o);
        if (serviceInfo == null) {
            throw new com.sun.star.uno.RuntimeException(
                "error: The object imported with " + args[0] + " did not support XServiceInfo", null);
```

```
        }

        if (!serviceInfo.supportsService("com.sun.star.io.Pipe")) {
            throw new com.sun.star.uno.RuntimeException(
                "error: The object imported with "+args[0]+" does not support the pipe service", null);
        }

        XOutputStream output = (XOutputStream) UnoRuntime.queryInterface(XOutputStream.class,o);
        XInputStream input = (XInputStream) UnoRuntime.queryInterface(XInputStream.class,o);

        // construct an array.
        byte[] array = new byte[]{1,2,3,4,5};

        // send it to the remote object
        output.writeBytes(array);
        output.closeOutput();

        // now read it again in two blocks
        byte [][] read = new byte[1][0];
        System.out.println("Available bytes : " + input.available());
        input.readBytes( read,2 );
        System.out.println("read " + read[0].length + ":" + read[0][0] + "," + read[0][1]);
        System.out.println("Available bytes : " + input.available());
        input.readBytes(read,3);
        System.out.println("read " + read[0].length + ":" + read[0][0] +
            "," + read[0][1] + "," + read[0][2]);

        System.out.println("Terminating client");
        System.exit(0);
    }
}
```

After bootstrapping the component context, the `UnoUrlResolver` service is instantiated to access remote objects. After resolving the remote object, check whether it really supports the `Pipe` service. For instance, try to connect this client to a running OpenOffice.org — this check will fail. A `byte array` with five elements is written to the remote server and read again with two `readBytes()` calls. Starting the client with the following command line connects to the server started above. You should get the following output:

```
I:\tmp>java UnoExeClient uno:socket,host=localhost,port=2002;urp;test
Available bytes : 5
read 2:1,2
Available bytes : 3
read 3:3,4,5
Terminating client
```

Using the uno Executable

The main benefit of using the *uno* tool as a replacement for writing executables is that the service manager initialization is separated from the task-solving code and the component can be reused. For example, to have multiple `XMain` implementations run in parallel in one process. There is more involved when writing a component compared to writing an executable. With the bootstrap variable mechanism there is a lot of freedom in bootstrapping the service manager (see chapter *3.4.2 Professional UNO - UNO Language Bindings - UNO C++ Binding*).

The *uno* tool is a good starting point when exporting a certain component as a remote server. However, when using the UNO technology later, the tool does have some disadvantages, such as multiple objects can not be exported or the component can only be initialized with command line arguments. If the *uno* tool becomes insufficient, the listening part in an executable will have to be re-implemented.



To instantiate Java components in build version 641, you need a complete setup so that the uno executable can find the java.ini file.

5 Advanced UNO

5.1 Choosing an Implementation Language

The UNO technology provides a framework for cross-platform and language independent programming. All the OpenOffice.org components can be implemented in any language supported by UNO, as long as they only communicate with other components through their IDL interfaces.



Note: The condition "as long as they only communicate with other components through their IDL interfaces" is to be strictly taken. In fact, a lot of implementations within OpenOffice.org export UNO interfaces and still use private C++ interfaces. This is a tribute to older implementations that cannot be rewritten in an acceptable timeframe.

A developer can customize the office to their needs with this flexibility, but they will have to decide which implementation language should be selected for a specific problem.

5.1.1 Supported Programming Environments

The support for programming languages in UNO and OpenOffice.org is divided into three different categories.

1) Languages that invoke calls on existing UNO objects are possibly implemented in other programming languages. Additionally, it may be possible to implement certain UNO interfaces, but not UNO components that can be instantiated by the service manager.
2) Languages that implement UNO components. UNO objects implemented in such a language are accessible from any other language that UNO supports, just by instantiating a service by name at the servicemanager. For instance, the developer can implement a OpenOffice.org Calc addin (see *8 Spreadsheet Documents*).
3) Languages that are used to write code to be delivered within OpenOffice.org documents and utilize dialogs designed with the OpenOffice.org dialog editor.

The following table lists programming languages currently supported by UNO. 'Yes' in the table columns denotes full support, 'no' denotes that there is no support and is not even planned in the future. 'Maybe in future' means there is currently no support, but this may change with future releases.

Language	UNO scripting	UNO components	Deployment with OpenOffice.org documents
C++	yes	yes	no
C	maybe in future	maybe in future	no
Java	yes	yes	maybe in future
StarBasic	yes	no	yes
OLE automation (win32 only)	yes	maybe in future	maybe in future
Python	maybe in future (under development)	maybe in future (under development)	maybe in future

Java

Java is a an accepted programming language offering a standard library with a large set of features and available extensions. Additional extensions will be available in the future, such as JAX-RPC for calling webservices. It is a typesafe language with a typesafe UNO binding. Although interfaces have to be queried explicitly, the type safety makes it suitable for larger projects. UNO components can be implemented with Java, that is, the Java VM is started on demand inside the office process when a Java component is instantiated. The OfficeBean allows embedding OpenOffice.org documents in Java Applets and Applications.

There is a constant runtime overhead of about 1 to 2 ms per call that is caused by the bridge conversion routines when calling UNO objects implemented in other language bindings. Since OpenOffice.org consists of C++ code, every Java call into the office needs to be bridged. This poses no problems if there are a few calls per user interaction. The runtime overhead will hurt the application when routines produce hundreds or thousands of calls.

C++

C++ is an accepted programming language offering third-party products. In addition to C++ being fast since it is compiled locally, it offers the fastest communication with OpenOffice.org because most of the essential parts of office have been developed in C++. This advantage becomes less important as you call into the office through the interprocess bridge, because every remote call means a constant loss of 1 to2 ms. The fastest code to extend the office can be implemented as a C++ UNO component. It is appropriate for larger projects due to its strong type safety at compile time.

C++ is difficult to learn and coding, in general, takes longer, for example, in Java. The components must be built for every platform, that leads to a higher level of complexity during development and deployment.

OpenOffice.org Basic

OpenOffice.org Basic is the scripting language developed for and integrated directly into OpenOffice.org. It currently offers the best integration with OpenOffice.org, because you can insert code into documents, attach arbitrary office events, such as document loading, keyboard shortcuts or menu entries, to Basic code and use dialogs designed within the OpenOffice.org IDE. In Basic, calls are invoked on an object rather than on a specific interface. Interfaces, such as

`com.sun.star.beans.XPropertySet` are integrated as Basic object properties. Basic always runs in the office process and thus avoids costly interprocess calls.

The language is type unsafe, that is, only a minimal number of errors are found during compilation. Most errors appear at runtime, therefore it is not the best choice for large projects. The language is OpenOffice.org specific and only offers a small set of runtime functionality with little third-party support. All office functionality is used through UNO. UNO components cannot be implemented with Basic. The only UNO objects that can be implemented are listeners. Finally, Basic does not offer any thread support.

OLE Automation Bridge

The OLE Automation bridge opens the UNO world to programming environments that support OLE automation, such as Visual Basic, JScript, Delphi or C++ Builder. Programmers working on the Windows platform can write programs for OpenOffice.org without leaving their language by learning a new API. These programmers have access to the libraries provided by their language. It is possible to implement UNO objects, if the programming language supports object implementation.

This bridge is only useful on a Win32 machine, thereby being a disadvantage . Scripts always run in a different process so that every UNO call has at least the usual interprocess overhead of 1 to 2 ms. Currently Automation UNO components cannot be implemented for the service manager, but this may change in the future.

Python

A Python scripting bridge (PyUNO) is currently developed by Ralph Thomas. It is available in an experimental alpha state with known limitations. For details, see PyUNO on *udk.openoffice.org*.

5.1.2 Use Cases

The following list gives typical UNO applications for the various language environments.

Java

- Servlets creating Office Documents on the fly, Java Server Pages
- Server-Based Collaboration Platforms, Document Management Systems
- Calc add-ins
- Chart add-ins
- Database Drivers

C++

- Filters reading document data and generating Office Documents through UNO calls
- Database Drivers
- Database Drivers

- Calc add-ins
- Chart add-ins

OpenOffice.org Basic

- Office Automation
- Event-driven data-aware forms

OLE Automation

- Office Automation, creating and controlling Office Documents from other applications and from Active Server Pages

Python

- Calc add-ins

5.1.3 Recommendation

All languages have their advantages and disadvantages as previously discussed , but there is not one language for all purposes, depending on your use. Consider carefully before starting a new project and evaluate the language to use so that it saves you time.

Sometimes it may be useful to use multiple languages to gain the advantages of both languages. For instance, currently it is not possible to attach a keyboard event to a java method, therefore, write a small basic function, which forwards the event to a java component.

The number of languages supported by UNO may increase and some of the limitations shown in the table above may disappear.

5.2 Language Bindings

UNO language bindings enable developers to use and implement UNO objects in arbitrary programming languages. Thus, the existing language bindings connect between implementation environments, such as Java, C++, OpenOffice.org Basic and OLE Automation. The connection is accomplished by *bridges*. The following terms are used in our discussion about the implementation of language bindings.

In our context, the *target language* or *target environment* denotes the language or environment from which the UNO component model is accessed. The *bridging language* is the language used for writing the bridge code.

An object-oriented language determines the layout of its objects in memory. We call an object that is based on this layout a *language object*. The layout along with everything that relates to it, such as creation, destruction, and interaction, is the *object model* of a language.

A *UNO proxy* (short: *proxy*) is created by a bridge and it is a language object that represents a UNO object in the target language. It provides the same functionality as the original UNO object. There

are two terms which further specialize a *UNO proxy*. The UNO *interface* proxy is a UNO proxy representing exactly *one* interface of a UNO object, whereas a UNO *object* proxy represents an uno object with *all* its interfaces.

An *interface bridge* bridges one UNO interface to one interface of the target language, that is, to a UNO interface proxy. When the proxy is queried for another interface that is implemented by the UNO object, then another interface proxy is returned. In contrast, an *object bridge* bridges entire UNO objects into UNO object proxies of the target language. The object proxy receives calls for all interfaces of the UNO object.

5.2.1 Implementing UNO Language Bindings

This section introduces the basic steps to create a new language binding. The steps required depend on the target language. The section provides an overview of existing language bindings to help you to decide what is necessary for your case. It is recommended that you read the sources for available language bindings and transfer the solutions they offer to the new circumstances of your target language.

Overview of Language Bindings and Bridges

Creating a language binding for UNO involves the following tasks:

Language Specification and UNO Feature Support

When writing a language binding, consider how to map UNOIDL types to your target language, treat simple types and handle complex types, such as `struct`, `sequence`, `interface` and `any`. Furthermore, UNOIDL features, such as services, properties and exceptions must be matched to the capabilities of the target language and accommodated, if need be.

Code Generator

If the target language requires type definitions at compile time, a code generator must translate UNOIDL type definitions to the target language type definitions according to the language specification, so that the types defined in UNOIDL can be used.

UNO Bridge

UNO communication is based on calls to interfaces. Bridges supply the necessary means to use interfaces of UNO objects between implementation environments. The key for bridging is an intermediate environment called binary UNO,that consists of binary data formats for parameters and return values, and a C dispatch method used to call arbitrary operations on UNO interfaces. A bridge must be capable of the following tasks:

- Between the target language and OpenOffice.org:
    - a) Converting operation parameters from the target language to binary UNO.
    - b) Transforming operation calls in the target language to calls in binary UNO in a different environment.
    - c) Transporting the operation call with its parameters to OpenOffice.org and the return values back to the target language.

    - d) Mapping return values from binary UNO to the target language.
- Between OpenOffice.org and the target language, that is, during callbacks or when using a component in the target language:
    - a) Converting operation parameters from binary UNO to the target language.
    - b) Transforming operation calls in binary UNO to calls in the target language.
    - c) Transporting the operation call with its parameters to the target language and the return values back to OpenOffice.org.
    - d) Converting return values from the target language to binary UNO.

The Reflection API delivers information about UNO types and is used by bridges to support type conversions (`com.sun.star.script.Converter`), and method invocations (`com.sun.star.script.Invocation` and `com.sun.star.script.XInvocation`). Furthermore, it supplies runtime type information and creates instances of certain UNO types, such as structs (`com.sun.star.reflection.CoreReflection`).

UNO Component Loader

An implementation loader is required to load and activate code produced by the target language if implementations in the target language are to be instantiated. This involves locating the component files produced by the target language, and mechanisms to load and execute the code produced by the target language, such as launching a runtime environment. Currently there are implementation loaders for jar files and locally shared libraries on the platforms supported by UNO.

Bootstrapping

A UNO language binding must prepare itself so that it can bridge to the UNO environments. It depends on the target environment how this is achieved. In Java, C++, and Python, a local service manager in the target environment is used to instantiate a com.sun.star.bridge.UnoUrlResolver that connects to OpenOffice.org. In the Automation bridge, the object com.sun.star.ServiceManager is obtained from the COM runtime system and in OpenOffice.org Basic the service manager is available from a special method of the Basic runtime environment, `getProcessServiceManager()`.

Implementation Options

There are two different approaches when creating a UNO language binding.

A) Programming languages checking types at compile time.
Examples are the languages Java or C++. In these environments, it is necessary to query for interfaces at certain objects and then invoke calls compile-time-typesafe on these interfaces.

B) Programming languages checking types at runtime.
Examples are the languages StarBasic, Python or Perl. In these languages, the interfaces are not queried explicitly as there is no compiler to check the signature of a certain method. Instead, methods are directly invoked on objects. During execution, the runtime engine checks if a method is available at one of the exported interfaces, and if not, a runtime error is raised. Typically, such a binding has a slight performance disadvantage compared to the solution above.

You can achieve different levels of integration with both types of language binding.

1) Call existing UNO interfaces implemented in different bindings.
   This is the normal scripting use case, for example, connect to a remote running office, instantiate some services and invoke calls on these services (*unidirectional binding*).

2) Implement UNO interfaces and let them be called from different bindings.
   In addition to 1) above, a language binding is able to implement UNO interfaces, for example, for instance listener interfaces, so that your code is notified of certain events (*limited bidirectional binding*).

3) Implement a UNO component that is instantiated on demand from any other language at the global service manager.
   In addition to 2) above, a binding must provide the code which starts up the runtime engine of the target environment. For example, when a Java UNO component is instantiated by the OpenOffice.org process, the Java VM must be loaded and initialized, before the actual component is loaded (*bidirectional binding*).

A language binding should always be bidirectional. That is, it should be possible to access UNO components implemented *in the target language* from OpenOffice.org, as well as accessing UNO components that are implemented in a different language *from the target language.*

The following table provides an overview about the capabilities of the different language bindings currently available for OpenOffice.org:

Language	**scripting (accessing office objects)**	**interface implementation**	**component development**
C++ (platform dependent)	yes	yes	yes
Java	yes	yes	yes
StarBasic	yes	(only listener interfaces)	no
OLE automation (Win32 only)	yes	yes	no (maybe in the future)

The next section outlines the implementation of a C++ language binding. The C++ binding itself is extremely platform and compiler dependent, which provides a barrierwhen porting OpenOffice.org to a new platform. Although this chapter focuses on C++ topics, the chapter can be applied for other typesafe languages that store their code in a shared library, for instance, Delphi, because the same concepts apply.

The section *5.2.3 Advanced UNO - Language Bindings - UNO Reflection API* considers the UNO reflection and invocation API, which offers generic functionality to inspect and call UNO objects. The section *5.2.4 Advanced UNO - Language Bindings - XInvocation Bridge* explains how the Reflection API is used to implement a runtime type-checking language binding.

The final chapter *5.2.5 Advanced UNO - Language Bindings - Implementation Loader* briefly describes the concept of *implementation loaders* that instantiates components on demand independently of the client and the implementation language.The integration of a new programming language into the UNO component framework is completed once you have a loader.

5.2.2 UNO C++ bridges

This chapter focuses on writing a UNO bridge locally, specifically writing a C++ UNO bridge to connect to code compiled with the C++ compiler. This is an introduction for bridge implementers.. It is assumed that the reader has a general understanding of compilers and a of 80x86 assembly language. Refer to the section *5.2.5 Advanced UNO - Language Bindings - Implementation Loader* for additional information.

Binary UNO Interfaces

A primary goal when using a new compiler is to adjust the C++-UNO data type generator (*cppumaker* tool) to produce binary compatible declarations for the target language. The tested cppu core functions can be used when there are similar sizes and alignment of UNO data types. The layout of C++ data types, as well as implementing C++-UNO objects is explained in *3.4.2 Professional UNO - UNO Language Bindings - UNO C++ Binding.*

When writing C++ UNO objects, you are implementing UNO interfaces by inheriting from pure virtual C++ classes, that is, the generated cppumaker classes (see *.hdl* files). When you provide an interface, you are providing a pure virtual class pointer. The following paragraph describes how the memory layout of a C++ object looks.

A C++-UNO interface pointer is always a pointer to a virtual function table (vftable), that is, a C++ `this` pointer. The equivalent binary UNO interface is a pointer to a struct `_uno_Interface` that contains function pointers. This struct holds a function pointer to a `uno_DispatchMethod()` and also a function pointer to `acquire()` and `release()`:

```
// forward declaration of uno_DispatchMethod()

typedef void (SAL_CALL * uno_DispatchMethod)(
         struct _uno_Interface * pUnoI,
         const struct _typelib_TypeDescription * pMemberType,
         void * pReturn,
         void * pArgs[],
         uno_Any ** ppException );

// Binary UNO interface

typedef struct _uno_Interface
{
         /** Acquires uno interface.

            @param pInterface uno interface
         */
         void (SAL_CALL * acquire )( struct _uno_Interface * pInterface );
         /** Releases uno interface.

            @param pInterface uno interface
         */
         void (SAL_CALL * release )( struct _uno_Interface * pInterface );
         /** dispatch function
         */
         uno_DispatchMethod pDispatcher ;
} uno_Interface;
```

Similar to `com.sun.star.uno.XInterface`, the life-cycle of an interface is controlled using the `acquire()` and `release()` functions of the binary UNO interface. Any other method is called through the dispatch function pointer `pDispatcher`. The dispatch function expects the binary UNO interface pointer (`this`), the interface member type of the function to be called, an optional pointer for a return value, the argument list and finally a pointer to signal an exception has occurred.

The caller of the dispatch function provides memory for the return value and the exception holder (`uno_Any`).

The `pArgs` array provides pointers to binary UNO values, for example, a pointer to an interface reference (`_uno_Interface **`) or a pointer to a SAL 32 bit integer (`sal_Int32 *`).

A bridge to binary UNO maps interfaces from C++ to binary UNO and conversely. To achieve this, implement a mechanism to produce proxy interfaces for both ends of the bridge.

C++ Proxy

A C++ interface proxy carries its interface type (reflection), as well as its destination binary UNO interface (`this` pointer). The proxy's vftable pointer is patched to a generated vftable that is capable of determining the index that was called ,as well as the `this` pointer of the proxy object to get the interface type.

The vftable requires an assembly code. The rest is programmed in C/C++. You are not allowed to trash the registers. On many compilers, the `this` pointer and parameters are provided through stack space. The following provides an example of a Visual C++ 80x86:

```
vftable slot0:
mov eax, 0
jmp cpp_vftable_call
vftable slot0:
mov eax, 1
jmp cpp_vftable_call
vftable slot0:
mov eax, 2
jmp cpp_vftable_call
...

static __declspec(naked) void __cdecl cpp_vftable_call(void)
{
__asm
        {
                sub             esp, 8          // space for immediate return type
                push            esp
                push            eax             // vtable index
                mov             eax, esp
                add             eax, 16
                push    eax                     // original stack ptr
                call            cpp_mediate     // proceed in C/C++
                add             esp, 12
                // depending on return value, fill registers
                cmp             eax, typelib_TypeClass_FLOAT
                je              Lfloat
                cmp             eax, typelib_TypeClass_DOUBLE
                je              Ldouble
                cmp             eax, typelib_TypeClass_HYPER
                je              Lhyper
                cmp             eax, typelib_TypeClass_UNSIGNED_HYPER
                je              Lhyper
                // rest is eax
                pop             eax
                add             esp, 4
                ret
Lhyper:         pop             eax
                pop             edx
                ret
Lfloat:         fld             dword ptr [esp]
                add             esp, 8
                ret
Ldouble:        fld             qword ptr [esp]
                add             esp, 8
                ret
        }
}
```

The vftable is filled with pointers to the different slot code (snippets). The snippet code recognizes the table index being called and calls `cpp_vftable_call()`. That function calls a C/C++ function (`cpp_mediate()`) and sets output registers upon return, for example, for floating point numbers depending on the return value type.

Remember that the vftable handling described above follows the Microsoft calling convention, that is, the `this` pointer is always the first parameter on the stack. This is currently not the case for gcc that prepends a pointer to a complex return value before the this pointer on the stack if a method returns a struct. This complicates the (static) vftable treatment, because different vftable slots have to be generated for different interface types, adjusting the offset to the proxy `this` pointer:

Microsoft Visual C++ call stack layout (esp offset [byte]):
0: return address
4: this pointer
8: optional pointer, if return value is complex (i.e. struct to be copy-constructed)
12: param0
16: param1
20: ...

This is usually the hardest part for stack-oriented compilers. Afterwards proceed in C/C++ (`cpp_mediate()`) to examine the proxy interface type, read out parameters from the stack and prepare the call on the binary UNO destination interface.

Each parameter is read from the stack and converted into binary UNO. Use cppu core functions if you have adjusted the *cppumaker* code generation (alignment, sizes) to the binary UNO layout (see *cppu/inc/uno/data.h*).

After calling the destination `uno_dispatch()` method, convert any `out/inout` and return the values back to C++-UNO, and return to the caller. If an exception is signalled (`*ppException != 0`), throw the exception provided to you in `ppException`. In most cases, you can utilize Runtime Type Information (RTTI) from your compiler framework to throw exceptions in a generic manner. Disassemble code throwing a C++ exception, and observe what the compiler generates.

Binary UNO Proxy

The proxy code is simple for binary UNO. Convert any `in/inout` parameters to C++-UNO values, preparing a call stack. Then perform a virtual function call that is similar to the following example for Microsoft Visual C++:

```
void callVirtualMethod(
         void * pThis, sal_Int32 nVtableIndex,
         void * pRegisterReturn, typelib_TypeClass eReturnTypeClass,
         sal_Int32 * pStackLongs, sal_Int32 nStackLongs )
{
         // parameter list is mixed list of * and values
         // reference parameters are pointers

__asm
         {
                  mov              eax, nStackLongs
                  test             eax, eax
                  je               Lcall
                  // copy values
                  mov              ecx, eax
                  shl              eax, 2                    // sizeof(sal_Int32) == 4
                  add              eax, pStackLongs          // params stack space
Lcopy:            sub              eax, 4
                  push             dword ptr [eax]
                  dec              ecx
                  jne              Lcopy
Lcall:
                  // call
                  mov              ecx, pThis
                  push             ecx                       // this ptr
                  mov              edx, [ecx]                // pvft
                  mov              eax, nVtableIndex
                  shl              eax, 2                    // sizeof(void *) == 4
                  add              edx, eax
                  call             [edx]                     // interface method call must be __cdecl!!!

                  // register return
                  mov              ecx, eReturnTypeClass
                  cmp              ecx, typelib_TypeClass_VOID
                  je               Lcleanup
                  mov              ebx, pRegisterReturn
// int32
                  cmp              ecx, typelib_TypeClass_LONG
                  je               Lint32
```

```
                cmp             ecx, typelib_TypeClass_UNSIGNED_LONG
                je              Lint32
                cmp             ecx, typelib_TypeClass_ENUM
                je              Lint32
// int8
                cmp             ecx, typelib_TypeClass_BOOLEAN
                je              Lint8
                cmp             ecx, typelib_TypeClass_BYTE
                je              Lint8
// int16
                cmp             ecx, typelib_TypeClass_CHAR
                je              Lint16
                cmp             ecx, typelib_TypeClass_SHORT
                je              Lint16
                cmp             ecx, typelib_TypeClass_UNSIGNED_SHORT
                je              Lint16
// float
                cmp             ecx, typelib_TypeClass_FLOAT
                je              Lfloat
// double
                cmp             ecx, typelib_TypeClass_DOUBLE
                je              Ldouble
// int64
                cmp             ecx, typelib_TypeClass_HYPER
                je              Lint64
                cmp             ecx, typelib_TypeClass_UNSIGNED_HYPER
                je              Lint64
                jmp             Lcleanup // no simple type
Lint8:
                mov             byte ptr [ebx], al
                jmp             Lcleanup
Lint16:
                mov             word ptr [ebx], ax
                jmp             Lcleanup
Lfloat:
                fstp            dword ptr [ebx]
                jmp             Lcleanup
Ldouble:
                fstp            qword ptr [ebx]
                jmp             Lcleanup
Lint64:
                mov             dword ptr [ebx], eax
                mov             dword ptr [ebx+4], edx
                jmp             Lcleanup
Lint32:
                mov             dword ptr [ebx], eax
                jmp             Lcleanup
Lcleanup:
                // cleanup stack
                mov             eax, nStackLongs
                shl             eax, 2                  // sizeof(sal_Int32) == 4
                add             eax, 4                  // this ptr
                add             esp, eax
        }
}
```

First stack data is pushed to the stack., including a `this` pointer, then the virtual function's pointer is retrieved and called. When the call returns, the return register values are copied back. It is also necessary to catch all exceptions generically and retrieve information about type and data of a thrown exception. In this case, look at your compiler framework functions also.

Additional Hints

Every local bridge is different, because of the compiler framework and code generation and register allocation. Before starting, look at your existing bridge code for the processor, compiler, and the platform in module *bridges/source/cpp_uno* that is part of the OpenOffice.org source tree on *www.openoffice.org*.

Also test your bridge code extensively and build the module *cppu* with debug symbols before implementing the bridge, because cppu contains alignment and size tests for the compiler.

For quick development, use the executable build in *cppu/test* raising your bridge library, doing lots of calls with all kinds of data on mapped interfaces.

Also test your bridge in a non-debug build. Often, bugs in assembly code only occur in non-debug versions, because of trashed registers. In most cases, optimized code allocates or uses more processor registers than non-optimized (debug) code.

5.2.3 UNO Reflection API

This section describes the UNO Reflection API. This API includes services and interfaces that can be used to get information about interfaces and objects at runtime.

XTypeProvider Interface

The interface `com.sun.star.lang.XTypeProvider` allows the developer to retrieve all types provided by an object. These types are usually interface types and the `XTypeProvider` interface can be used at runtime to detect which interfaces are supported by an object. This interface should be supported by every object to make it scriptable from OpenOffice.org Basic.

Converter Service

The service `com.sun.star.script.Converter` supporting the interface `com.sun.star.script.XTypeConverter` provides basic functionality that is important in the reflection context. It converts values to a particular type. For the method `com.sun.star.script.XTypeConverter:convertTo()`, the target type is specified as `type`, allowing any type available in the UNO type system. The method `com.sun.star.script.XTypeConverter:convertToSimpleType()` converts a value into a simple type that is specified by the corresponding `com.sun.star.uno.TypeClass`. If the requested conversion is not feasible, both methods throw a `com.sun.star.script.CannotConvertException`.

CoreReflection Service

The service `com.sun.star.reflection.CoreReflection` supporting the interface `com.sun.star.reflection.XIdlReflection` is an important entry point for the Uno Reflection API. The `XIdlReflection` interface has two methods that each return a `com.sun.star.reflection.XIdlClass` interface for a given name (method `forName()`) or `any` value (method `getType()`).

The interface `XIdlClass` is one of the central interfaces of the Reflection API. It provides information about types, especially about class or interface, and struct types. Besides general information, for example, to check type identity through the method `equals()` or to determine a type or class name by means of the method `getName()`, it is possible to ask for the fields or members, and methods supported by an interface type (method `getFields()` returning a sequence of `XIdlField` interfaces and method `getMethods()` returning a sequence of `XIdlMethod` interfaces).

The interface `XIdlField` is deprecated and should not be used. Instead the interface `com.sun.star.reflection.XIdlField2` is available by querying it from an `XIdlField` interface returned by an `XIdlClass` method.

The interface `XIdlField` or `XIdlField2` represents a struct member of a struct or get or set accessor methods of an interface type. It provides information about the field (methods `getType()`

and `getAccessMode()`) and reads and – if allowed by the access mode – modifies its value for a given instance of the corresponding type (methods `get()` and `set()`).

The interface `XIdlMethod` represents a method of an interface type. It provides information about the method (methods `getReturnType()`, `getParameterTypes()`, `getParameterInfos()`, `getExceptionTypes()` and `getMode()`) and invokes the method for a given instance of the corresponding type (method `invoke()`).

Introspection

The service `com.sun.star.beans.Introspection` supporting the interface `com.sun.star.beans.XIntrospection` is used to inspect an object of interface or struct type to obtain information about its members and methods. Unlike the `CoreReflection` service, and the `XIdlClass` interface ,the inspection is not limited to one interface type but to all interfaces supported by an object. To detect the interfaces supported by an object, the `Introspection` service queries for the `XTypeProvider` interface. If an object does not support this interface, the introspection does not work correctly.

To inspect an object, pass it as an `any` value to the `inspect()` method of `XIntrospection`. The result of the introspection process is returned as `com.sun.star.beans.XIntrospectionAccess` interface. This interface is used to obtain information about the inspected object. All information returned refers to the complete object as a combination of several interfaces. When accessing an object through `XIntrospectionAccess`, it is impossible to distinguish between the different interfaces.

The `com.sun.star.beans.XIntrospectionAccess` interface provides a list of all properties (method `getProperties()`) and methods (method `getMethods()`) supported by the object. The introspection maps methods matching the pattern

```
FooType getFoo()
 setFoo(FooType)
```

to a property `Foo` of type `FooType`.

`com.sun.star.beans.XIntrospectionAccess` also supports a categorization of properties and methods. For instance, it is possible to exclude "dangerous" methods ,such as the reference counting methods `com.sun.star.uno.XInterface:acquire()` and `com.sun.star.uno.XInterface:release()` from the set of methods returned by `getMethods()`. When the `Introspection` service is used to bind a new scripting language, it is useful to block the access to functionality that could crash the entire OpenOffice.org application when used in an incorrect manner.

The `XIntrospectionAccess` interface does not allow the developer to invoke methods and access properties directly. To invoke methods, the `invoke()` method of the `XIdlMethod` interfaces returned by the methods `getMethods()` and `getMethod()` are used. To access properties, a `com.sun.star.beans.XPropertySet` interface is used that can be queried from the `com.sun.star.beans.XIntrospectionAccess:queryAdapter()` method. This method also provides adapter interfaces for other generic access interfaces like `com.sun.star.container.XNameAccess` and `com.sun.star.container.XIndexAccess`, if these interfaces are also supported by the original object.

Invocation

The service `com.sun.star.script.Invocation` supporting the interface `com.sun.star.lang.XSingleServiceFactory` provides a generic, high-level access (higher compared to the `Introspection` service) to the properties and methods of an object. The object that should be accessed through `Introspection` is passed to the

com.sun.star.lang.XSingleServiceFactory:createInstanceWithArguments() method. The returned XInterface can then be queried for com.sun.star.script.XInvocation2 derived from com.sun.star.script.XInvocation.

The XInvocation interface invokes methods and access properties directly by passing their names and additional parameters to the corresponding methods (method invoke(), getValue() and setValue()). It is also possible to ask if a method or property exists with the methods hasMethod() and hasProperty().

When invoking a method with invoke(), the parameters are passed as a sequence of any values. The Invocation service automatically converts these arguments, if possible to the appropriate target types using the com.sun.star.script.Converter service that is further described below. The Introspection functionality is suitable for binding scripting languages to UNO that are not or only weakly typed.

The XInvocation2 interface extends the Invocation functionality by methods to ask for further information about the properties and methods of the object represented by the Invocation instance. It is possible to ask for the names of all the properties and methods (method getMemberNames()) and detailed information about them represented by the com.sun.star.script.InvocationInfo struct type (methods getInfo() and getInfoForName()).

Members of struct com.sun.star.script.InvocationInfo	
aName	Name of the method or property.
eMemberType	Kind of the member (method or property).
PropertyAttribute	Only for property members: This field may contain zero or more constants of the com::sun::star::beans::PropertyAttribute constants group. It is not guaranteed that all necessary constants are set to describe the property completely, but a flag will be set if the corresponding characteristic really exists. For example,iIf the READONLY flag is set, the property is read only. If it is not set, the property nevertheless can be read only. This field is irrelevant for methods and is set to 0.
aType	Type of the member, when referring to methods, the return type
aParamTypes	Types of method parameters, for properties this sequence is empty
aParamModes	Mode of method parameters (in, out, inout), for properties this sequence is empty.

The Invocation service is based on the Introspection service. The XInvocation interface has a method getIntrospection() to ask for the corresponding XIntrospectionAccess interface. The Invocation implementation currently implemented in OpenOffice.org supports this, but in general, an implementation of XInvocation does not provide access to an XInvocationAccess interface.

InvocationAdapterFactory

The service com.sun.star.script.InvocationAdapterFactory supporting the interfaces com.sun.star.script.XInvocationAdapterFactory and com.sun.star.script.XInvocationAdapterFactory2 are used to create adapters that map a generic XInvocation interface to specific interfaces. This functionality is especially essential for creating scripting language bindings that do not only access UNO from the scripting language, but also to implement UNO objects using the scripting language. Without the InvocationAdapterFactory functionality, this would only be possible if the scripting language supported the implementation of interfaces directly.

By means of the `InvocationAdapterFactory` functionality it is only necessary to map the scripting language specific native invocation interface, for example, realized by an OLE `IDispatch` interface, to the UNO `XInvocation` interface. Then, any combination of interfaces needed to represent the services supported by a UNO object are provided as an adapter using the `com.sun.star.script.XInvocationAdapterFactory2:createAdapter()` method.

Another important use of the invocation adapter is to create listener interfaces that are passed to the corresponding `add...Listener()` method of an UNO interface and maps to the methods of an interface to `XInvocation`. In this case, usually the `com.sun.star.script.XInvocationAdapterFactory:createAdapter()` method is used.

XTypeDescription

Internally, types in UNO are represented by the type `type`. This type also has an interface representation `com.sun.star.reflection.XTypeDescription`. A number of interfaces derived from `XTypeDescription` represent types. These interfaces are:

- `com.sun.star.reflection.XArrayTypeDescription`
- `com.sun.star.reflection.XCompoundTypeDescription`
- `com.sun.star.reflection.XEnumTypeDescription`
- `com.sun.star.reflection.XIndirectTypeDescription`
- `com.sun.star.reflection.XUnionTypeDescription`
- `com.sun.star.reflection.XInterfaceTypeDescription`
- `com.sun.star.reflection.XInterfaceAttributeTypeDescription`
- `com.sun.star.reflection.XInterfaceMemberTypeDescription`
- `com.sun.star.reflection.XInterfaceMethodTypeDescription`

The corresponding services are `com.sun.star.reflection.TypeDescriptionManager` and `com.sun.star.reflection.TypeDescriptionProvider`. These services support `com.sun.star.container.XHierarchicalNameAccess` and asks for a type description interface by passing the fully qualified type name to the `com.sun.star.container.XHierarchicalNameAccess:getByHierarchicalName()` method.

The `TypeDescription` services and interfaces are listed here for completeness. Ordinarily this functionality would not be used when binding a scripting language to UNO, because the high-level services `Invocation`, `Introspection` and `Reflection` provide all the functionality required. If the binding is implemented in C++, the type `type` and the corresponding C API are used directly.

The following illustration provides an overview of how the described services and interfaces work together. Each arrow expresses a "uses" relationship. The interfaces listed for a service are not necessarily supported by the service directly, but contain interfaces that are strongly related to the services.

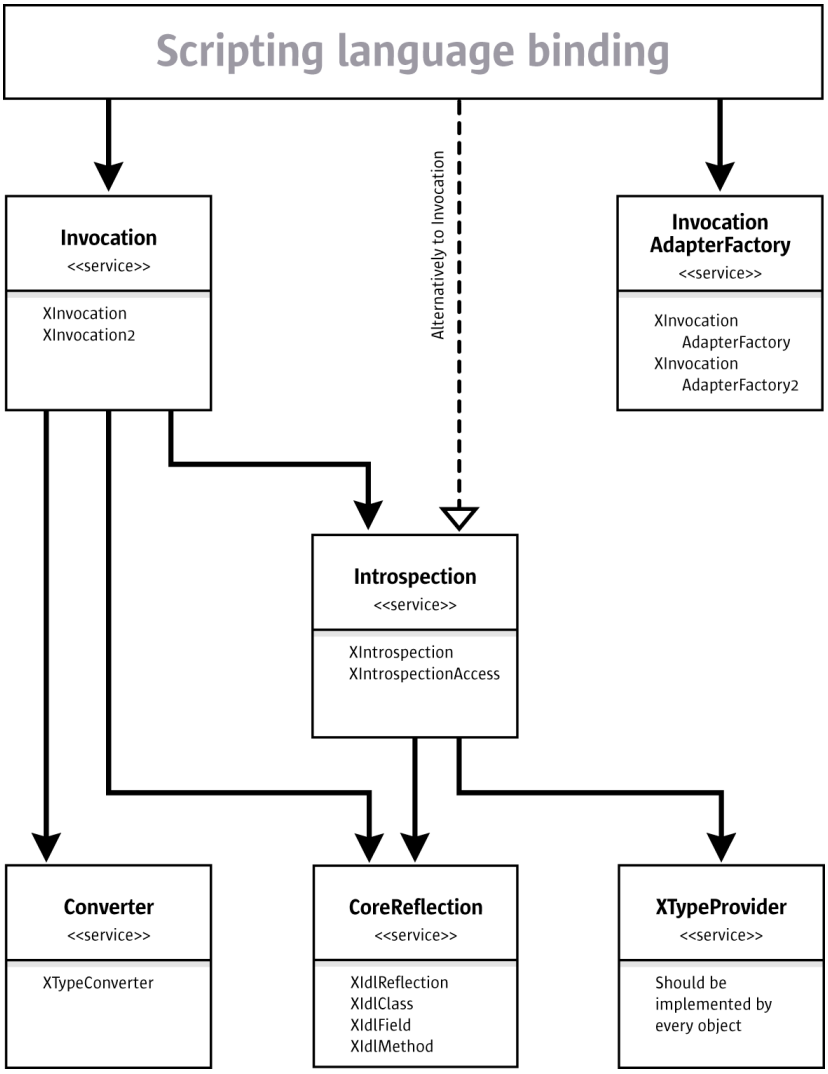


Illustration 44

5.2.4 XInvocation Bridge

Scripting Existing UNO Objects

This section describes UNO bridges for type-unsafe (scripting) programming languages. These bridges are based on the `com.sun.star.script.Invocation` service.

The most common starting point for a new scripting language binding is that you want to control OpenOffice.org from a script running externally. To accomplish this, you need to know what your scripting language offers to extend the language, for example, Python or Perl extend the language with a module concept using locally shared libraries.

In general, your bridge must offer a static method that is called from a script. Within this method, bootstrap a UNO C++ component context as described in *[Chapter:Components.Deployment.Bootstrapping]*.

Proxying a UNO Object

Next, this component context must be passed to the script programmer, so that you can instantiate a `com.sun.star.bridge.UnoUrlResolver` and connect to a running office within the script.

The component context can not be passed directly as a C++ UNO reference, because the scripting engine does not recognize it, therefore build a language dependent proxy object around the C++ object Reference.

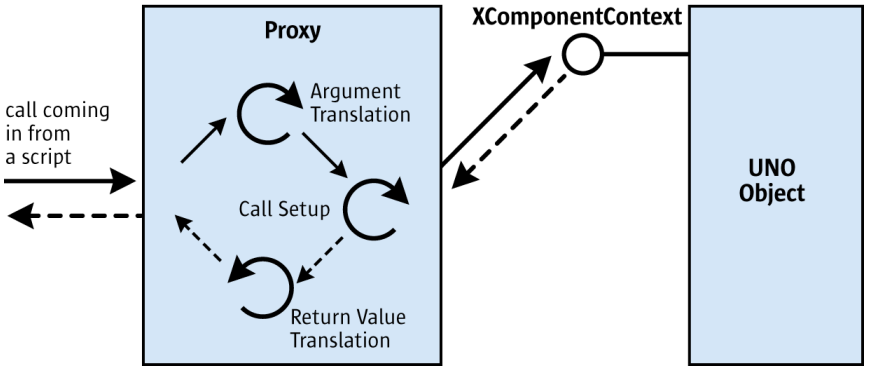


Illustration 45

For example, Python offers an API to create a proxy. Typically calls invoked on the proxy from a script are narrowed into one single C function. The Python runtime passes method names and an array containing the arguments to this C function.

If a proxy is implemented for a concrete interface, the method names that you received could in theory be compared to all method names offered by the UNO interface. This is not feasible, because of all the interfaces used in OpenOffice.org. The `com.sun.star.script.Invocation` service exists for this purpose. It offers a simple interface `com.sun.star.lang.XSingleServiceFactory` that creates a proxy for an arbitrary UNO object using the `createInstanceWithArguments()` method and passing the object the proxy acts for. Use the `com.sun.star.script.XInvocation` interface that is exported by this proxy to invoke a method on the UNO object.

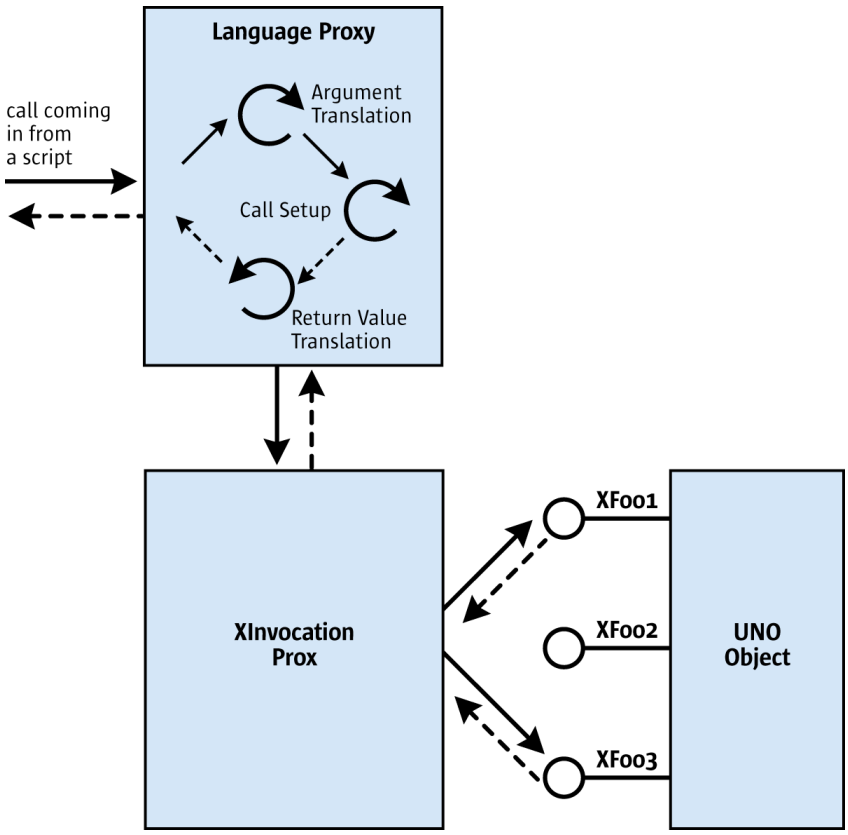


Illustration 46

Argument Conversion

In addition, argument conversion must be c onsidered by specifying how each UNO type should be mapped to your target language.

Convert the language dependent data types to UNO data types before calling `com.sun.star.script.XInvocation:invoke()` and convert the UNO datatypes (return value and out parameters) to language dependent types after the call has been exectuted. The conversion routines are typically recursive functions, because data values are nested in complex types, such as `struct` or `any`.

When UNO object references are returned by method calls to UNO objects, create new language dependent proxies as discussed above. When passing a previously returned UNO object as a parameter to a new method call, the language binding must recognize that it is a proxied object and pass the original UNO object reference to the `com.sun.star.script.XInvocation:invoke()` call instead.

A special case for conversions are UNOIDL structs. You want to call a method that takes a struct as an argument. The first problem is the struct must be created by the bridge and the script programmer must be able to set members at the struct. One solution is that the bridge implementer creates a UNO struct using core C functions from the `cppu` library, but this is complicated and results in a lot of difficulty.

Therefore, a solution has been created that accesses structs through the `XInvocation` interface, as if they were UNO objects. This simplifies struct handling for bridge programmers. Refer to the reference documentation of `com.sun.star.reflection.CoreReflection` and the `com.sun.star.script.Invocation` service and the `com.sun.star.beans.XMaterialHolder` interface.

Exception Handling

UNO method calls may throw exceptions and must be mapped to the desired target language appropriately, depending on the capabilities of your target language. Ideally, the target language supports an exception concept, but error handlers, such as in OpenOffice.org Basic can be used also. A third way and worst case scenario is to check after every API call if an exception has been thrown,. In case the UNO object throws an excception, the `XInvocation` proxy throws a `com.sun.star.reflection.InvocationTargetException`. The exception has an additional `any` member, that contains the exception that was really thrown.

Note that the `XInvocation` proxy may throw a `com.sun.star.script.CannotConvertException` indicating that the arguments passed by the script programmer cannot be matched to the arguments of the desired function. For example, there are missing arguments or the types are incompatible. This must be reported as an error to the script programmer.

Property Support

The `com.sun.star.script.Invocation` has special `getProperty()` and `setProperty()` methods. These methods are used when the UNO object supports a property set and your target language, for example, supports something similar to the following:

```
object.propname = 'foo';.
```

Note that every property is also reachable by `com.sun.star.script.XInvocation:invoke ('setPropertyValue', ...)`, so these set or getProperty functions are optional.

Implementing UNO objects

When it is possible to implement classes in your target language, consider offering support for implementation of UNO objects. This is useful for callbacks, for example, event listeners. Another typical use case is to provide a datasource through a `com.sun.star.io.XInputStream`.

The script programmer determines which UNOIDL types the developed class implements, such as flagged by a special member name, for example, such as `__supportedUnoTypes`.

When an instance of a class is passed as an argument to a call on an external UNO object, the bridge code creates a new language dependent proxy that additionally supports the `XInvocation` interface. the bridge code hands the `XInvocation` reference of the bridge's proxy to the called object. This works as long as the `com.sun.star.script.XInvocation:invoke()` method is used directly, for instance OpenOffice.org Basic, except if the called object expects an `XInputStream`.

The `com.sun.star.script.InvocationAdapterFactory` service helps by creating a proxy for a certain object that implements `XInvocation` and a set of interfaces, for example, given by the `__supportedUnoTypes` variable. The proxy returned by the `createAdapater()` method must be passed to the called object instead of the bridge's `XInvocation` implementation. When the `Adapter` is queried for one of the supported types, an appropriate proxy supporting that interface is created.

If a UNO object invokes a call on the object, the bridge proxy's `com.sun.star.script.XInvocation:invoke()` method is called. It converts the passed argu-

ments from UNO types to language dependent types and conversely using the same routines you have for the other calling direction. Finally, it delegates the call to the implementation within the script.

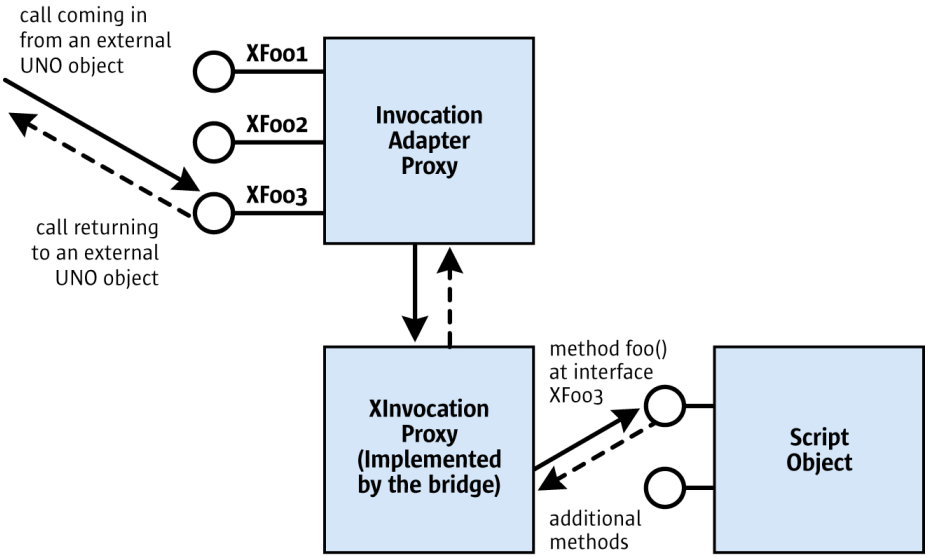


Illustration 47

It may become difficult if you do not want to start with an external scripting engine, but want to use the scripting engine inside the OpenOffice.org process instead. This must be supported by the target language. Often it is possible to load some library dynamically and access the scripting runtime engine through a C API. It should be implemented as a UNO C++ component. There are currently no generic UNO interfaces for this case, except for the `com.sun.star.loader.XImplementationLoader`. Define your own interfaces that best match your requirements. You might instantiate from Basic and retrieve an initial object or start a script. Future versions of OpenOffice.org may have a more comprehensive solution.

Example: Python Bridge PyUNO

This section provides an example of how the Python UNO bridge PyUNO bootstraps a service manager and how it makes use of the `Invocation` service to realize method invocation. While some parts are implementation or Python specific, the example provides a general understanding of language bindings.

The Python bridge PyUNO uses the cppu helper library to bootstrap a local service manager that is asked for a `UnoUrlResolver` service in Python.

In *UNO.py*, Python calls `PyUNO.bootstrap()` and receives a local component context. Note the parameter setup in that, it points to an ini file that configures the bootstrapped service manager with a type library. The file *setup.ini* corresponds to the *uno.ini* file that is used with the global service manager of the office.

```
import PyUNO
import os

setup_ini = 'file:///%s/setup.ini' % os.getenv ('PWD')

class UNO:

        def __init__ ( self, connection='socket,host=localhost,port=2002;urp', setup=setup_ini ):
                """ do the bootstrap

                        connection can be one or more of the following:
```

```
                        socket,
                        host = localhost | <hostname> | <ip-addr>,
                        port = <port>,
                        service = soffice,
                        user = <username>,
                        password = <password>
                        ;urp

                """

                self.XComponentContext = PyUNO.bootstrap ( setup )
                self.XUnoUrlResolver, o = \
                        self.XComponentContext.ServiceManager.createInstanceWithContext (
                                'com.sun.star.bridge.UnoUrlResolver', self.XComponentContext )
                self.XNamingService, o = self.XUnoUrlResolver.resolve (
                                'uno:%s;StarOffice.NamingService' % connection )
                self.XMultiServiceFactory, o = self.XNamingService.getRegisteredObject (
                                'StarOffice.ServiceManager')
                self.XComponentLoader, o = \
                        self.XMultiServiceFactory.createInstance ( 'com.sun.star.frame.Desktop' )
                ...
```

Python uses function tables to map Python to C functions. *PyUNO_module.cc* defines a table with the mappings for the PyUNO object. As shown in the following example, `PyUNO.bootstrap()` is mapped to the C function `newBootstrapPyUNO()`:

```
static struct PyMethodDef PyUNOModule_methods [] =
  {
    {"bootstrapPyUNO", bootstrapPyUNO, 1},
    {"bootstrap ", newBootstrapPyUNO , 1},
    {"createIdlStruct", createIdlStruct, 1},
    {"true", createTrueBool, 1},
    {"false", createFalseBool, 1},
    {NULL, NULL}
  };
```

The function `newBootstrapPyUNO()` calls `Util::bootstrap()` in *PyUNO_Util.cc* and passes the location of the *setup.ini* file.

```
static PyObject* newBootstrapPyUNO (PyObject* self, PyObject* args)
{
  char* ini_file_location;
  Reference<XComponentContext> tmp_cc;
  Any a;

  if (!PyArg_ParseTuple (args, "s", &ini_file_location))
    return NULL;
  tmp_cc = Util::bootstrap (ini_file_location);
  ...
```

`Util::bootstrap()` uses `defaultBootstrap_InitialComponentContext(iniFile)` from *cppuhelper/bootstrap.hxx* to create a local component context and its parameter `iniFile` points to the *setup.ini* file that configures the local service manager to use *applicat.rdb*. This local component context instantiates services, such as the `UnoUrlResolver`.

```
  Reference<XComponentContext> bootstrap (char* ini_file_location)
  {
    Reference<XComponentContext> my_component_context;
    try
      {
          my_component_context = defaultBootstrap_InitialComponentContext (
                  OUString::createFromAscii (ini_file_location ));
      }
    catch (com::sun::star::uno::Exception e)
      {
          printf (OUStringToOString (e.Message, osl_getThreadTextEncoding ()).getStr ());
      }
    return my_component_context;
  }
```

Now `newBootstrapPyUNO()` continues to set up a UNO proxy. It creates local instances of `com.sun.star.script.Invocation` and `com.sun.star.script.Converter`, and calls `PyUNO_new()`, passing the local `ComponentContext`, a reference to the `XSingleServiceFactory` interface of `com.sun.star.script.Invocation` and a reference to the `XTypeConverter` interface of `com.sun.star.script.Converter`.

```
static PyObject* newBootstrapPyUNO (PyObject* self, PyObject* args)
{
  char* ini_file_location;
```

```
  Reference<XComponentContext> tmp_cc;
  Any a;

  if (!PyArg_ParseTuple (args, "s", &ini_file_location))
    return NULL;
  tmp_cc = Util::bootstrap (ini_file_location) ;
  Reference<XMultiServiceFactory> tmp_msf (tmp_cc->getServiceManager (), UNO_QUERY);
  if (!tmp_msf.is ())
    {
      PyErr_SetString (PyExc_RuntimeError, "Couldn't bootstrap from inifile");
      return NULL;
    }
  Reference<XSingleServiceFactory> tmp_ssf (tmp_msf->createInstance (
         OUString (RTL_CONSTASCII_USTRINGPARAM ("com.sun.star.script.Invocation "))), UNO_QUERY);
  Reference<XTypeConverter> tmp_tc (tmp_msf->createInstance (
         OUString (RTL_CONSTASCII_USTRINGPARAM ("com.sun.star.script.Converter "))), UNO_QUERY);
  if (!tmp_tc.is ())
    {
      PyErr_SetString (PyExc_RuntimeError, "Couldn't create XTypeConverter");
      return NULL;
    }
  if (!tmp_ssf.is ())
    {
      PyErr_SetString (PyExc_RuntimeError, "Couldn't create XInvocation");
      return NULL;
    }
  a <<= tmp_cc;

  return PyUNO_new (a, tmp_ssf, tmp_tc) ;

}
```

`PyUNO_new()` in *PyUNO.cc* is the function responsible for building all Python proxies. The call to `PyUNO_new()` here in `newBootstrapPyUno()` builds the first local PyUNO proxy for the ComponentContext object `a` which has been returned by `Util::bootstrap()`.

For this purpose, `PyUNO_new()` uses the `Invocation` service to retrieve an `XInvocation2` interface to the `ComponentContext` service passed in the parameter `a`:

```
// PyUNO_new
//
// creates Python object proxies for the given target UNO interfaces
//    targetInterface     given UNO interface
//    ssf                 XSingleServiceFactory interface of com.sun.star.script.Invocation service
//    tc                  XTypeConverter interface of com.sun.star.script.Converter service

PyObject* PyUNO_new (Any targetInterface,
                 Reference<XSingleServiceFactory> ssf,
                 Reference<XTypeConverter> tc)
  ...
  Sequence<Any> arguments (1);
  Reference<XInterface> tmp_interface;
  ...
  // put the target object into a sequence of Any for the call to
  // ssf->createInstanceWithArguments()
  // ssf is the XSingleServiceFactory interface of the com.sun.star.script.Invocation service
  arguments[0] <<= targetInterface;

  // obtain com.sun.star.script.XInvocation2 for target object from Invocation
  // let Invocation create an XInvocation object for the Any in arguments
  tmp_interface = ssf->createInstanceWithArguments (arguments);
  // query XInvocation2 interface
  Reference<XInvocation2 > tmp_invocation (tmp_interface, UNO_QUERY);
  ...
```

The Python proxy invokes methods, and creates and converts UNO types. This Python specific and involves the implementation of several functions according to the Python API.

Finally `__init__()` in *UNO.py* in the above example uses the PyUNO object to obtain a local `UnoUrlResolver` that retrieves the initial object from the office.

5.2.5 Implementation Loader

When you are raising a service by name using the `com.sun.star.lang.ServiceManager` service, the service manager decides an implementation name, code location and an appropriate loader to

raise the code. It is commonly reading out of a persistent registry storage, for example, *applicat.rdb*, for this purpose. Previously, the *regcomp* tool has registered components into that registry during the OpenOffice.org setup. The tool uses a service called `com.sun.star.registry.ImplementationRegistration` for this task.

A loader knows how to load a component from a shared library, a .jar or script file and is able to obtain the service object factory for an implementation and retrieve information being written to the registry. A specific loader defines how a component implementer has to package code so that it is recognized by UNO. For instance in C++, a component is a shared library and in Java it is a .jar file. In a yet to be developed loader, the implementer of the loader has to decide, what a component is in that particular language – it might as well be a single script file.

The `com.sun.star.loader.XImplementationLoader` interface looks like the following:

```
interface XImplementationLoader: com::sun::star::uno::XInterface
{

        com::sun::star::uno::XInterface activate ( [in] string implementationName,
                        [in] string implementationLoaderUrl,
                        [in] string locationUrl,
                        [in] com::sun::star::registry::XRegistryKey xKey )
                        raises( com::sun::star::loader::CannotActivateFactoryException );

        boolean writeRegistryInfo ( [in] com::sun::star::registry::XRegistryKey xKey,
                        [in] string implementationLoaderUrl,
                        [in] string locationUrl )
                        raises( com::sun::star::registry::CannotRegisterImplementationException );
};
```

The `locationUrl` argument describes the location of the implementation file,for example, a jar file or a shared library. The `implementationLoaderUrl` argument is not used and is obsolete. The registry key `xKey` writes information about the implementations within a component into a persistent storage. Refer to *4.6.4 Writing UNO Components - C++ Component - Write Registration Info Using Helper Method* for additional information.

The method `writeRegistryMethod()` is called by the *regcomp* tool to register a component into a registry.

The `activate()` method returns a factory `com.sun.star.lang.XSingleComponentFactory` for a concrete implementation name.

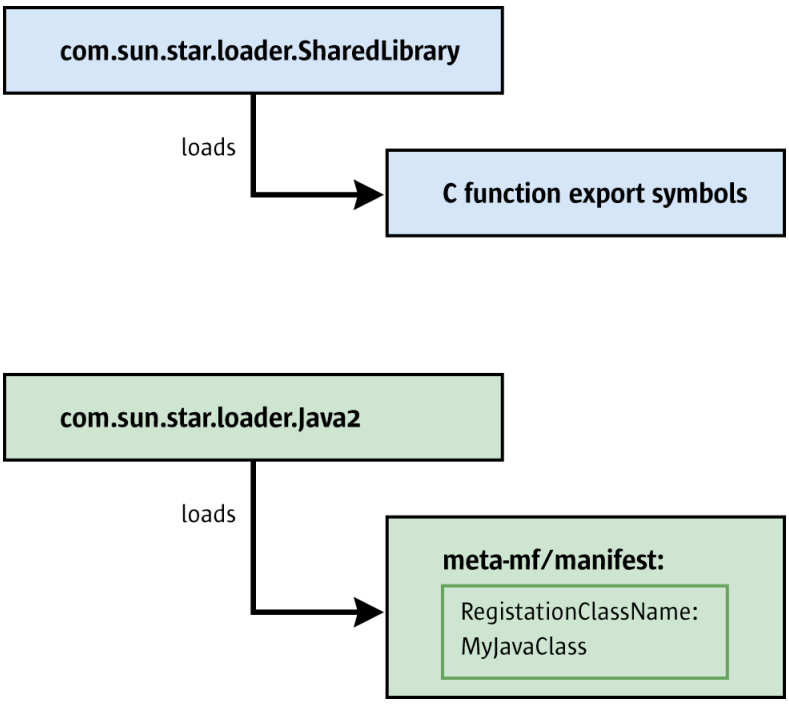


Illustration 48

The loader is often implemented in C/C++. When the loader is instantiated, it is responsible for starting up the language runtime, for example, Java VM, Python interpreter, through implementation. After starting up the runtime, the loader starts up the UNO language binding as discussed in the previous chapter, and bridge the `XRegistryKey` interface and the initial factory interface.

Shared Library Loader

This section discusses the loader for local components written in C++ that are loaded by the `com.sun.star.loader.SharedLibrary` service. Every type safe programming language that stores its code in shared libraries should implement the bridge with environments and mappings as discussed in chapters *5.2.1 Advanced UNO - Language Bindings - Implementing UNO Language Bindings - Overview of Language Bindings and Bridges - UNO Bridge* and *5.2.2 Advanced UNO - Language Bindings - UNO C++ Bridges*. These programming languages can reuse the existing loader without creating a new one.

When the shared library is mapped into the running process, for example, using `osl_loadModule()`, the shared library loader retrieves defined C symbols out of the library to determine the compiler that built the code . This function symbol is called `component_getImplementationEnvironment()`. When the code is compiled with the Microsoft Visual C++ compiler, it sets a pointer to a string called `"msci"`, with gcc 3.0.1 a string `"gcc3"` which is a UNO environment type name. A UNO environment is connected with the code that runs in it, for example, the code compiled with gcc3 runs in the UNO environment with type name gcc3.

In addition to the environment type name, a UNO environment defines a context pointer. The context pointer and environment type name define a unique UNO environment. Although the context pointer is mostly null, it is required to identify the environments apart for the same type, for example, to identify different Java virtual machine environments when running a UNO object in two different Java virtual machines within the same process. Both environments have the same type name `"java"`, but different context pointers. In local (C++) code, the context pointer is irrelevant , that is, set to null.The type name determines the UNO runtime environment.

When the loader knows the environment the code comes from, it decides if bridging is required. Bridging is needed if the loader code is compiled with a different compiler, thus running in a different environment. In this case, the loader raises a bridge to speak UNO with the component code.

The loader calls on two more functions related to the above `XimplementationLoader` interface. All of these symbols are C functions and have the following signatures:

```
extern "C" void SAL_CALL component_getImplementationEnvironment(
         const sal_Char ** ppEnvTypeName, uno_Environment ** ppEnv );
extern "C" sal_Bool SAL_CALL component_writeInfo(
         void * pServiceManager, void * pRegistryKey );
extern "C" void SAL_CALL component_getFactory(
         const sal_Char * pImplName, void * pServiceManager, void * pRegistryKey );
```

The latter two functions expect incoming C++-UNO interfaces, therefore the loader needs to bridge interfaces before calling the functions as stated above.

Bridges

The loader uses the cppu core runtime to map an interface, specifying the UNO runtime environment that needs the interface mapping. The cppu core runtime raises and connects the appropriate bridges, and provides a unidirectional mapping that uses underlying bidirectional bridges. Under Unix, the name of the bridge library follows the naming convention *lib<SourceEnvironment>_<TargetEnvironment>.,* Under Windows, *<SourceEnvironment>_<TargetEnvironment>.dll* is used. For instance, *libgcc3_uno.so* is the bridge library for mappings from gcc3 to binary UNO,and *msci_uno.dll* maps from MS Visual C++ to binary UNO. The bridges mentioned above all bridge to binary UNO. Binary UNO is only used as an intermediate environment. In general, do not program binary UNO in clients. The purpose is to reduce the number of necessary bridge implementations. New bridges have to map only to binary UNO instead of all conceivable bridge combinations.

5.2.6 Help with New Language Bindings

Every UNO language binding is different, therefore only most important points were stressed, that is, those that are likely to appear in almost every language binding implementation. Object issues, such as lifetime, object identity, `any` handling, and bootstrapping were not discussed, because they are too language dependent. For more information on these issues, subscribe to the *dev@udk.openoffice.org* mailing list to discuss these issues for your programming language.

5.3 Differences Between UNO and Corba

This subsection discusses the differences between UNO and CORBA by providing the fundamental differences and if the different concepts could be mapped into the world of the other model. Consider the following feature comparison. The column titled "Mapping possible" states if a feature could be mapped by a (yet to be developed) generic bridge.

	UNO	**CORBA**	**Mapping possible**
multiple inheritance of interfaces	no	yes	yes

	UNO	CORBA	Mapping possible
inheritance of structs	yes	no	yes
inheritance of exceptions	yes	no	yes
mandatory base interface for all interfaces	yes	no	yes
mandatory base exception for all exceptions	yes	no	yes
context of methods	no	yes	no
char	no	yes	yes
8 bit string	no	yes	yes
array	no	yes	yes
union	no	yes	yes
assigned values for enum	yes	no	yes
meta type 'type'	yes	no	yes
object identity	yes	no	no
lifetime mechanism	yes	no	no
succession of oneway calls	yes	no	no
in process communication	yes	no	no
thread identity	yes	no	no
customized calls	no	yes	yes
less code generation	yes	no	no

- Multiple Inheritance
  CORBA supports multiple inheritance of interfaces, whereas UNO only supports single inheritance.
  Mapping: Generates an additional interface with all methods and attributes of the inherited interfaces that must be implemented in addition to the other interfaces.

- Inheritance of Structs
  In contrast to CORBA, UNO supports inheritance of struct types. This is useful to define general types and more detailed subtypes.
  Mapping: Generate a struct with all members, plus all members of the inherited structs.

- Inheritance of Exceptions
  CORBA does not support inheritance for exceptions, whereas UNO does. Inheritance of exceptions allows the specification of a complex exception concept. It is possible to make fine granular concepts using the detailed exceptions in the layer where they are useful and the base exception in higher levels. The UNO error handling is based on exceptions and with inheritance of exceptions it is possible to specify 'error classes' with a base exception and more detailed errors of the same 'error class' that inherit from this base exception. On higher level APIs it is enough to declare the base exception to specify the 'error class' and it is possible to support all errors of this 'error class'.
  Mapping: Generates an exception with all members, plus all members of the inherited exceptions. This is the same solution as for structs.

- Mandatory Base Interfaces
  UNO specifies a mandatory base interface for all interfaces. This interface provides `acquire()` and `release()` functions for reference counting. The minimum life time of an object is managed by means of reference counting.

- Mandatory Base Exception
  UNO specifies a mandatory base exception for all exceptions. This base exception contains a string member `Message`that describes the reason for the exception in readable format. The base exception makes it also possible to catch all UNO exceptions separately.
- Method Context
  CORBA supports a request context. This context consists of a name-value pair which is specified for methods in UNOIDL. The context is used for describing the current state of the caller object. A request context provides additional, operation-specific information that may affect the performance of a request.
- Type `char`
  UNO does not support 8-bit characters. In UNO, `char` represents a 16-bit unicode character. Mapping: To support 8-bit characters it is possible to expand the `TypeClass` enum to support 8-bit characters and strings. The internal representation does not change anything, the `TypeClass` is only relevant for mapping.
- 8 bit string
  UNO does not support 8-bit strings. In UNO, `string` represents a 16-bit unicode string. Mapping: The same possibility as for `char`.
- Type `array`
  UNO does not support arrays at the moment, but is planned for the future.
- Type `union`
  UNO does not support unions at the moment, but is planned for the future.
- Assigned Values for `enums`
  UNO supports the assignment of values for enum values in IDL. This means that it is possible to use these values directly to specify or operate with the required enum value in target languages supporting this feature, for example, . C, C++.
  Mapping: Possible by using the names of the values.

5.4 UNO Design Patterns and Coding Styles

This chapter discusses design patterns and coding recommendations for OpenOffice.org. Possible candidates are:

- Singleton: global service manager, Desktop, UCB
- Factory: decouple specification and implementation, cross-environment instantiation, context-specific instances
- Listener: eliminate polling
- Element access: it is arguable if that is a design pattern or just an API
- Properties: solves remote batch access, but incurs the problem of compile-time type indifference
- UCB commands: universal dispatching of content specific operations
- Dispatch commands: universal dispatching of object specific operations, chain of responsibility

5.4.1 Double-Checked Locking

The double-checked locking idiom is sometimes used in C/C++ code to speed up creation of a single-instance resource. In a multi-threaded environment, typical C++ code that creates a single-instance resource might look like the following example:

```
#include "osl/mutex.hxx"

T * getInstance1()
{
    static T * pInstance = 0;
    ::osl::MutexGuard aGuard(::osl::Mutex::getGlobalMutex());
    if (pInstance == 0)
    {
        static T aInstance;
        pInstance = &aInstance;
    }
    return pInstance;
}
```

A mutex guards against multiple threads simultaneously updating `pInstance`, and the nested static `aInstance` is guaranteed to be created only when first needed, and destroyed when the program terminates.

The disadvantage of the above function is that it must acquire and release the mutex every time it is called. The double-checked locking idiom was developed to reduce the need for locking, leading to the following modified function. Do not use.:

```
#include "osl/mutex.hxx"

T * getInstance2()
{
    static T * pInstance = 0;
    if (pInstance == 0)
    {
        ::osl::MutexGuard aGuard(::osl::Mutex::getGlobalMutex());
        if (pInstance == 0)
        {
            static T aInstance;
            pInstance = &aInstance;
        }
    }
    return pInstance;
}
```

This version needs to acquire and release the mutex only when `pInstance` has not yet been initialized, resulting in a possible performance improvement. The mutex is still needed to avoid race conditions when multiple threads simultaneously see that `pInstance` is not yet initialized, and all want to update it at the same time. The problem with `getInstance2` is that it does not work.

Assume that thread 1 calls `getInstance2` first, finding `pInstance` uninitialized. It acquires the mutex, creates `aInstance` that results in writing data into `aInstance`'s memory, updates `pInstance` that results in writing data into `pIntance`'s memory, and releases the mutex. Some hardware memory models a write the operations that transfer `aInstance`'s and `pInstance`'s data to main memory to be re-ordered by the processor executing thread 1. Now, if thread 2 enters `getInstance2` when `pInstance`'s data has already been written to main memory by thread 1, but `aInstance`'s data has not been written yet (remember that write operations may be done out of order), then thread 2 sees that `pInstance` has already been initialized and exits from `getInstance2` directly. Thread 2 dereferences `pInstance` thereafter, accessing `aInstance`'s memory that has not yet been written into. Anything may happen in this situation.

In Java, double-checked locking can never be used, because it is broken and cannot be fixed.

In C and C++, the problem can be solved, but only by using platform-specific instructions, typically some sort of memory-barrier instructions. There is a macro `OSL_DOUBLE_CHECKED_LOCKING_MEMORY_BARRIER` in *osl/doublecheckedlocking.h* that uses the double-checked locking idiom in a way that actually works in C and C++.

```
#include "osl/doublecheckedlocking.h"
```

```
#include "osl/mutex.hxx"

T * getInstance3()
{
    static T * p = 0;
    T * pInstance = p;
    if (p == 0)
    {
        ::osl::MutexGuard aGuard(osl::Mutex::getGlobalMutex());
        p = pInstance;
        if (p == 0)
        {
            static T aInstance;
            p = &aInstance;
            OSL_DOUBLE_CHECKED_LOCKING_MEMORY_BARRIER();
            pInstance = p;
        }
    }
    else
        OSL_DOUBLE_CHECKED_LOCKING_MEMORY_BARRIER();
    return p;
}
```

The first (inner) use of `OSL_DOUBLE_CHECKED_LOCKING_MEMORY_BARRIER` ensures that `aInstance`'s data has been written to main memory before `pInstance`'s data is written, therefore a thread can not see `pInstance` to be initialized when `aInstance`'s data has not yet reached main memory. This solves the problem described above.

The second (outer) usage of `OSL_DOUBLE_CHECKED_LOCKING_MEMORY_BARRIER` is required to solve a problem concerning the reordering on Alpha processors.



For more information about this problem, see *Reordering on an Alpha processor* by Bill Pugh (*www.cs.umd.edu/~pugh/java/memoryModel/AlphaReordering.html*) and *Pattern-Oriented Software Architecture, Volume 2: Patterns for Concurrent and Networked Objects* by Douglas C. Schmidt et al (Wiley, 2000). Also see the Usenet article *Re:Talking about volatile and threads synchronization* by Davide Butenhof (October 2002) on why the outer barrier can be moved into an `else` clause.

If you are coding in C++, there is an easier way to use double-checked locking without worrying about the fine points. Use the `rtl_Instance` template from *rtl/instance.hxx*:

```
#include "osl/getglobalmutex.hxx"
#include "osl/mutex.hxx"
#include "rtl/instance.hxx"

namespace {
    struct Init()
    {
        T * operator()()
        {
            static T aInstance;
            return &aInstance;
        }
    };
}

T * getInstance4()
{
    return rtl_Instance< T, Init, ::osl::MutexGuard, ::osl::GetGlobalMutex >::create(
        Init(), ::osl::GetGlobalMutex());
}
```

Note that an extra function class is required in this case. The documentation of `rtl_Instance` contains further examples of how this template can be used.



If you are looking for more general information, the article *The "Double-Checked Locking is Broken" Declaration* (*http://www.cs.umd.edu/~pugh/java/memoryModel/DoubleCheckedLocking.html*) is a good source on double-checked locking, while *Computer Architecture: A Quantitative Approach*, Third Edition by John L. Hennessy and David A. Patterson (Morgan Kaufmann, 2002) and *UNIX® Systems for Modern Architectures: Symmetric Multiprocessing and Caching for Kernel Programmers* by Curt Schimmel (Addison-Wesley, 1994) offer detailed information about hardware memory models.

6 Office Development

This chapter describes the application environment of the OpenOffice.org application. It assumes that you have read the chapter *2 First Steps*, and that you are able to connect to the office and load documents.

In most cases, developers use the functionality of OpenOffice.org by opening and modifying documents. The interfaces and services common to all document types and how documents are embedded in the surrounding application environment are discussed.

It is also possible to extend the functionality of OpenOffice.org by replacing the services mentioned here by intercepting the communication between objects or by creating your own document type and integrating it into the desktop environment. All these things are discussed in this chapter.

6.1 OpenOffice.org Application Environment

6.1.1 Overview

The OpenOffice.org application environment is made up of the *desktop environment* and the *framework API*.

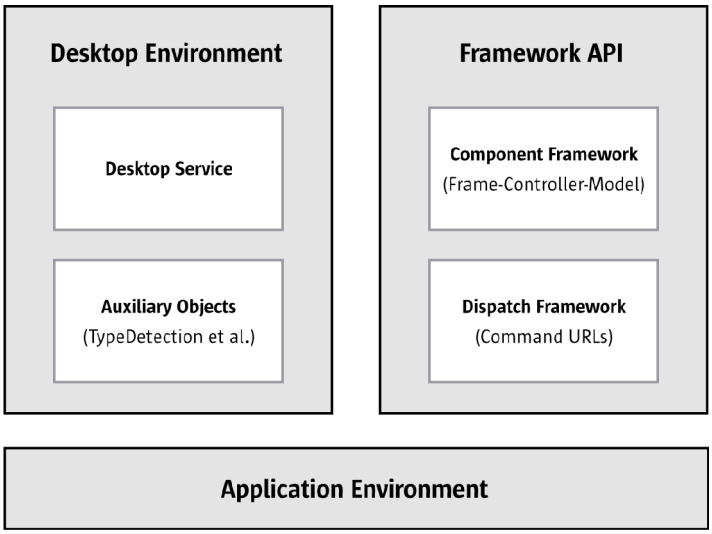


Illustration 49: OpenOffice.org Application Environment

The desktop environment consists of the desktop and auxiliary objects. It employs the framework API to carry out its functions. The framework API currently has two parts: the *component framework* and *dispatch framework*. The component framework follows a special Frame-Controller-Model paradigm to manage components viewable in OpenOffice.org. The dispatch framework handles command requests sent by the GUI.

Desktop Environment

The `com.sun.star.frame.Desktop` service is the central management instance for the OpenOffice.org application framework. All OpenOffice.org application windows are organized in a hierarchy of frames that contain viewable components. The desktop is the root frame for this hierarchy. From the desktop you can load viewable components, access frames and components, terminate the office, traverse the frame hierarchy and dispatch command requests.

The name of this service originates at StarOffice 5.x, where all document windows were embedded into a common application window that was occupied by the StarOffice desktop, mirroring the Windows desktop. The root frame of this hierarchy was called the desktop frame. The name of this service and the interface name `com.sun.star.frame.XDesktop` were kept for compatibility reasons.

The desktop object and frame objects use auxiliary services, such as the `com.sun.star.document.TypeDetection` service and other, opaque implementations that interact with the UNO-based office, but are not accessible through the OpenOffice.org API. Examples for the latter are the global document event handling and its user interface (**Tools – Configure – Events**), and the menu bars that use the dispatch API without being UNO services themselves. The desktop service, together with these surrounding objects, is called the *desktop environment*.

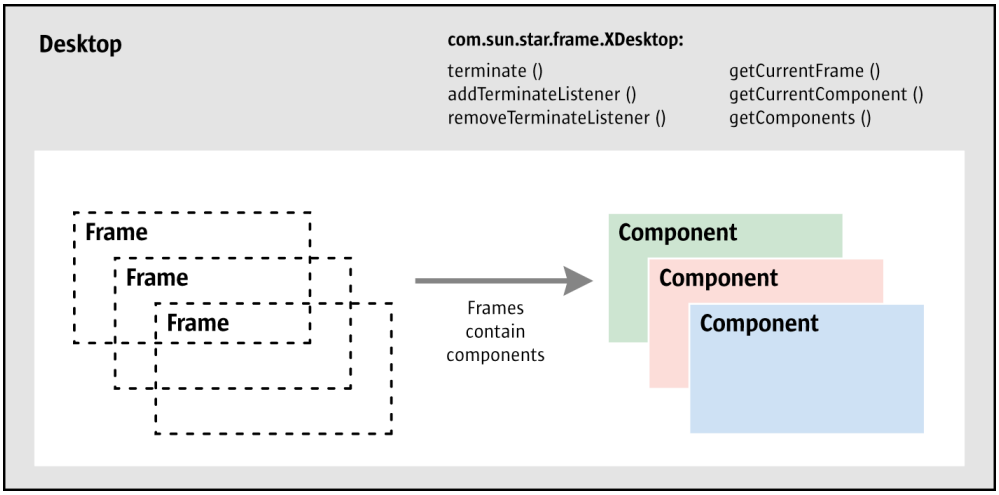


Illustration 50: The Desktop terminates the office and manages components and frames

The viewable components managed by the desktop can be three different kinds of objects: full-blown office documents with a document model and controllers, components with a controller but no model, such as the bibliography and database browser, or simple windows without API-enabled controllers, for example, preview windows. The commonality between these types of components is the `com.sun.star.lang.XComponent` interface. Components with controllers are also called *office components*, whereas simple window components are called *trivial components*.

Frames in the OpenOffice.org API are the connecting link between windows, components and the desktop environment. The relationship between frames and components are discussed in the next section *6.1.1 Office Development - OpenOffice.org Application Environment - Overview - Framework API*.

Like all other services, the `com.sun.star.frame.Desktop` service can be exchanged by another implementation that extends the functionality of OpenOffice.org. By exchanging the desktop service it is possible to use different kinds of windows or to make OpenOffice.org use MDI instead of SDI. This is not an easy thing to do, but it is possible without changing any code elsewhere in OpenOffice.org.

Framework API

The framework API does not define an all-in-one framework with strongly coupled interfaces, but defines specialized frameworks that are grouped together by implementing the relevant interfaces at OpenOffice.org components. Each framework concentrates on a particular aspect, so that each component decides the frameworks it wants to participate in.

Currently, there are two of these frameworks: the *component framework* that implements the frame-controller-model paradigm and the *dispatch framework* that handles command requests from and to the application environment. The controller and frame implementations form the bridge between the two frameworks, because controllers and frames implement interfaces from the component framework and dispatch framework.

The framework API is an abstract specification. Its current implementation uses the Abstract Window Toolkit (AWT) specified in `com.sun.star.awt`, which is an abstract specification as well. The current implementation of the AWT is the Visual Component Library (VCL), a cross-platform toolkit for windows and controls written in C++ created before the specification of com.sun.star.awt and adapted to support com.sun.star.awt.

Frame-Controller-Model Paradigm in OpenOffice.org

The well known Model-View-Controller (MVC) paradigm separates three application areas: document data (*model*), presentation (*view*) and interaction (*controller*). OpenOffice.org has a similar abstraction, called the Frame-Controller-Model (FCM) paradigm. The FCM paradigm shares certain aspects with MVC , but it has different purposes, therefore it is best to approach FCM independently from MVC. The model and controller in MVC and FCM are quite different things.

The FCM paradigm in OpenOffice.org separates three application areas: document object (*model*), screen interaction with the model (*controller*) and controller-window linkage (*frame*).

- The model holds the document data and has methods to change these data without using a controller object. Text, drawings, and spreadsheet cells are accessed directly at the model.
- The controller has knowledge about the current view status of the document and manipulates the screen presentation of the document, but not the document data. It observes changes made to the model, and can be duplicated to have multiple controllers for the same model.
- The frame contains the controller for a model and *knows* the windows that are used with it, but does not have window functionality.

The purpose of FCM is to have three exchangeable parts that are used with an exchangeable window system:

It is possible to write a new controller that presents an existing model in a different manner without changing the model or the frame. A controller depends on the model it presents, therefore a new controller for a *new* model can be written.

Developers can introduce new models for new document types without taking care of the frame and underlying window management system. However, since there is no default controller, it is necessary to write a suitable controller also.

By keeping all window-related functionality separate from the frame, it is possible to use one single frame implementation for every possible window in the entire OpenOffice.org application. Thus, the presentation of all visible components is customized by exchanging the frame implementation. At runtime you can access a frame and replace the controller, together with the model it controls, by a different controller instance.

Frames

Linking Components and Windows

The main role of a frame in the Frame-Controller-Model paradigm is to act as a liaison between viewable components and the window system.

Frames can hold one component, or a component and one or more subframes. The following diagrams: Illustration 49: OpenOffice.org Application Environment and Illustration 52: Frame containing a component and a sub-frame depict both possibilities. The first illustration 49 shows a frame containing only a component. It is connected with two window instances: the container window and component window.

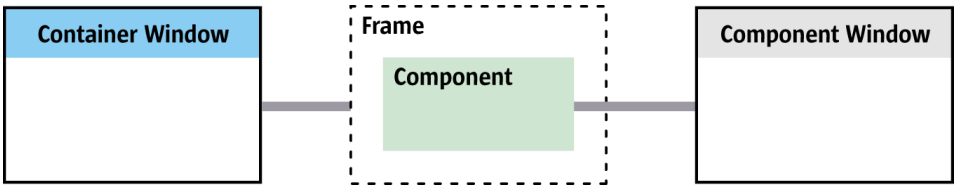


Illustration 51: Frame containing a component

When a frame is constructed, the frame must be initialized with a container window using `com.sun.star.frame.XFrame:initialize()`. This method expects the `com.sun.star.awt.XWindow` interface of a surrounding window instance, which becomes the container window of the frame. The window instance passed to `initialize()` must also support `com.sun.star.awt.XTopWindow` to become a container window. The container window must broadcast window events, such as `windowActivated()`, and appear in front of other windows or be sent to the background. The fact that container windows support `com.sun.star.awt.XTopWindow` does not mean the container window is an independent window of the underlying window system with a title bar and a system menu. An `XTopWindow` acts as a window if necessary, but it can also be docked or depend on a surrounding application window.

After initializing the frame, a component is set into the frame by a frame loader implementation that loads a component into the frame. It calls `com.sun.star.frame.XFrame:setComponent()` that takes another `com.sun.star.awt.XWindow` instance and the `com.sun.star.frame.XController` interface of a controller.Usually the controller is holding a model, therefore the component gets a component window of its own, separate from the container window.

A frame with a component is associated with *two* windows: the *container* window which is an `XTopWindow` and the *component* window, which is the rectangular area that displays the component and receives GUI events for the component while it is active. When a frame is initialized with an instance of a window in a call to `initialize()`, this window becomes its container window. When a component is set into a frame using `setComponent()`, another `com.sun.star.awt.XWindow` instance is passed becoming the component window.

When a frame is added to the desktop frame hierarchy, the desktop becomes the parent frame of our frame. For this purpose, the `com.sun.star.frame.XFramesSupplier` interface of the desktop is passed to the method `setCreator()` at the `XFrame` interface. This happens internally when the method `append()` is called at the `com.sun.star.frame.XFrames` interface supplied by the desktop.



A component window can have sub-windows, and that is the case with all documents in OpenOffice.org. For instance, a text document has sub-windows for the toolbars and the editable text. Form controls are sub-windows, as well, however, these sub-windows depend on the component window and do not appear in the Frame-Controller-Model paradigm, as discussed above.

The second diagram shows a frame with a component and a sub-frame with another component. Each frame has a container window and component window.

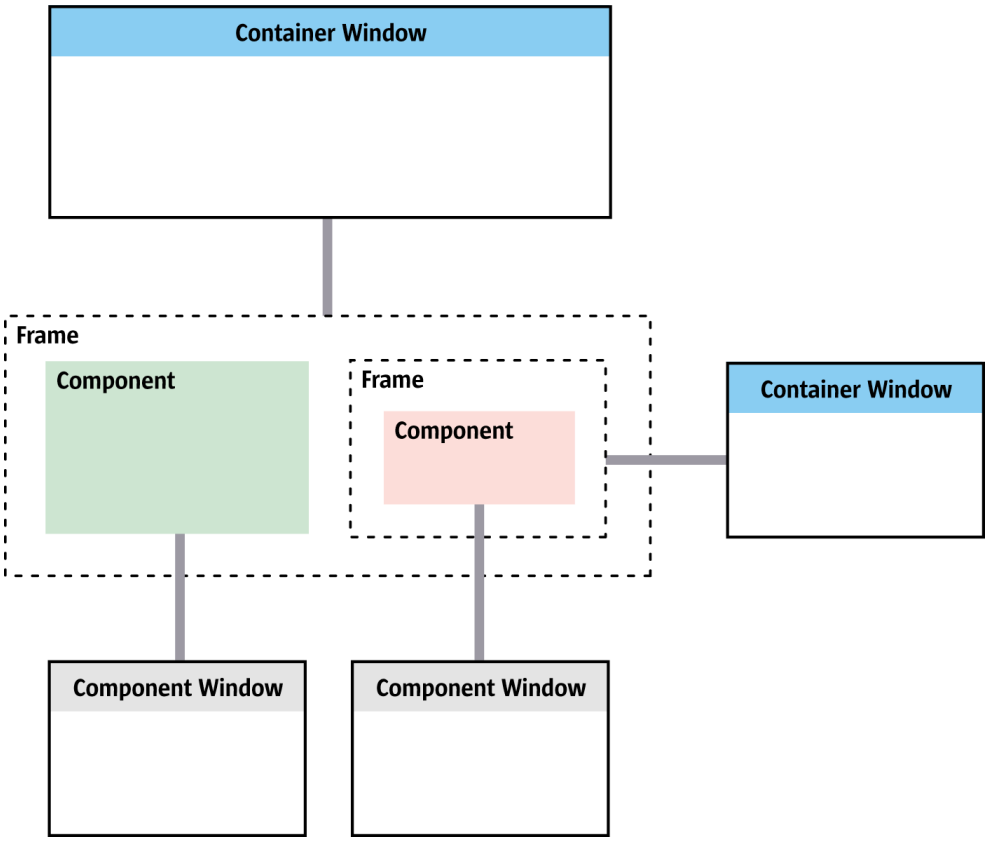


Illustration 52: Frame containing a component and a sub-frame

In the OpenOffice.org GUI, sub-frames appear as dependent windows. The sub-frame in Illustration 52 could be a dockable window, such as the beamer showing the database browser or a floating frame in a document created with **Insert – Frame**.

Note that a frame with a component and sub-frame is associated with *four* windows. The frame and the sub-frame have a container window and a component window for the component.

When a sub-frame is added to a surrounding frame, the frame becomes the parent of the sub-frame by a call to `setCreator()` at the sub-frame. This happens internally when the method `append()` is called at the `com.sun.star.frame.XFrames` interface supplied by the surrounding frame.

The section *6.1.4 Office Development - OpenOffice.org Application Environment - Creating Frames Manually* shows examples for the usage of the `XFrame` interface that creates frames in the desktop environment, constructs dockable and standalone windows, and inserts components into frames.

Communication through Dispatch Framework

Besides the main role of frames as expressed in the `com.sun.star.frame.XFrame` interface, frames play another role by providing a communication context for the component they contain, that is, every communication from a controller to the desktop environment, and the user interface and conversely is done through the frame. This aspect of a frame is published through the `com.sun.star.frame.XDispatchProvider` interface, that uses special command requests to trigger actions.

The section *6.1.6 Office Development - OpenOffice.org Application Environment - Using the Dispatch Framework* discusses the usage of the dispatch API.

Components in Frames

The desktop environment section discussed the three kinds of viewable components that can be inserted into a frame. If the component has a controller *and* a model like a document, or if it has only a controller, such as the bibliography and database browser, it implements the `com.sun.star.frame.Controller` service represented by the interface `com.sun.star.frame.XController`. In the call to `com.sun.star.frame.XFrame:setComponent()`, the controller is passed with the component window instance. If the component has no controller, it directly implements `com.sun.star.lang.XComponent` and `com.sun.star.awt.XWindow`. In this case, the component is passed as `XWindow` parameter, and the `XController` parameter must be an `XController` reference set to `null`.

If the viewable component is a *trivial component* (implementing `XWindow` only), the frame holds a reference to the component window, controls the lifetime of the component and propagates certain events from the container window to the component window. If the viewable component is an *office component* (having a controller), the frame adds to these basic functions a set of features for integration of the component into the environment by supporting additional command URLs for the component at its `com.sun.star.frame.XDispatchProvider` interface.

Controllers

Controllers in OpenOffice.org are between a frame and document model. This is their basic role as expressed in `com.sun.star.frame.XController`, which has methods `getModel()` and `getFrame()`. The method `getFrame()` provides the frame the controller is attached to. The method `getModel()` returns a document model, but it may return an empty reference if the component does not have a model.

Usually the controller objects support additional interfaces specific to the document type they control, such as `com.sun.star.sheet.XSpreadsheetView` for Calc document controllers or `com.sun.star.text.XTextViewCursorSupplier` for Writer document controllers.

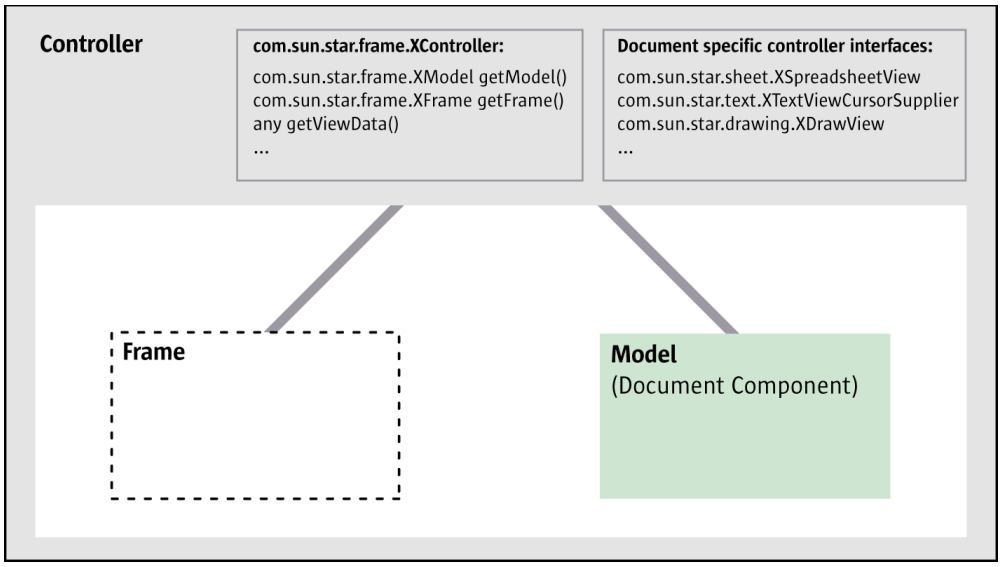


Illustration 53: Controller with Model and Frame

There can be more than one controller instance with frames of their own controlling the same document model simultaneously. Multiple controllers and frames are created by OpenOffice.org when the user clicks **Window – New Window**.

Windows

Windows in the OpenOffice.org API are rectangular areas that are positioned and resized, and inform listeners about UI events (`com.sun.star.awt.XWindow`). They have a platform-specific counterpart that is wrapped in the `com.sun.star.awt.XWindowPeer` interface, which is invalidated (redrawn), and sets the system pointer and hands out the toolkit for the window. The usage of the window interfaces is outlined in the section *6.1.3 Office Development - OpenOffice.org Application Environment - Using the Component Framework - Window Interfaces* below.

Dispatch Framework

The dispatch framework is designed to provide a uniform access to components for a GUI by using command URLs that mirror menu items, such as **Edit – Select All** with various document components. Only the component knows how to execute a command. Similarly, different document components trigger changes in the UI by common commands. For example, a controller might create UI elements like a menu bar, or open a hyperlink.

Command dispatching follows a chain of responsibility. Calls to the dispatch API are moderated by the frame, so all dispatch API calls from the UI to the component and conversely are handled by the frame. The frame passes on the command until an object is found that can handle it. It is possible to restrict, extend or redirect commands at the frame through a different frame implementation or through other components connecting to the frame.

It has already been discussed that frames and controllers have an interface `com.sun.star.frame.XDispatchProvider`. The interface is used to query a dispatch object for a command URL from a frame and have the dispatch object execute the command. This interface is one element of the dispatch framework.

By offering the interception of dispatches through the interface `com.sun.star.frame.XDispatchProviderInterception`, the `Frame` service offers a method to modify a component's handling of GUI event s while keeping its whole API available simultaneously.



Normally, command URL dispatches go to a target frame which decides what to do with it. A component can use globally accessible objects like the desktop service to bypass restrictions set by a frame, but this is not recommended. It is impossible to prevent a implemention of components against the design principles, because the framework API is made for components that adhere to its design.

The usage of the Dispatch Framework is described in the section *6.1.6 Office Development - OpenOffice.org Application Environment - Using the Dispatch Framework*

6.1.2 Using the Desktop

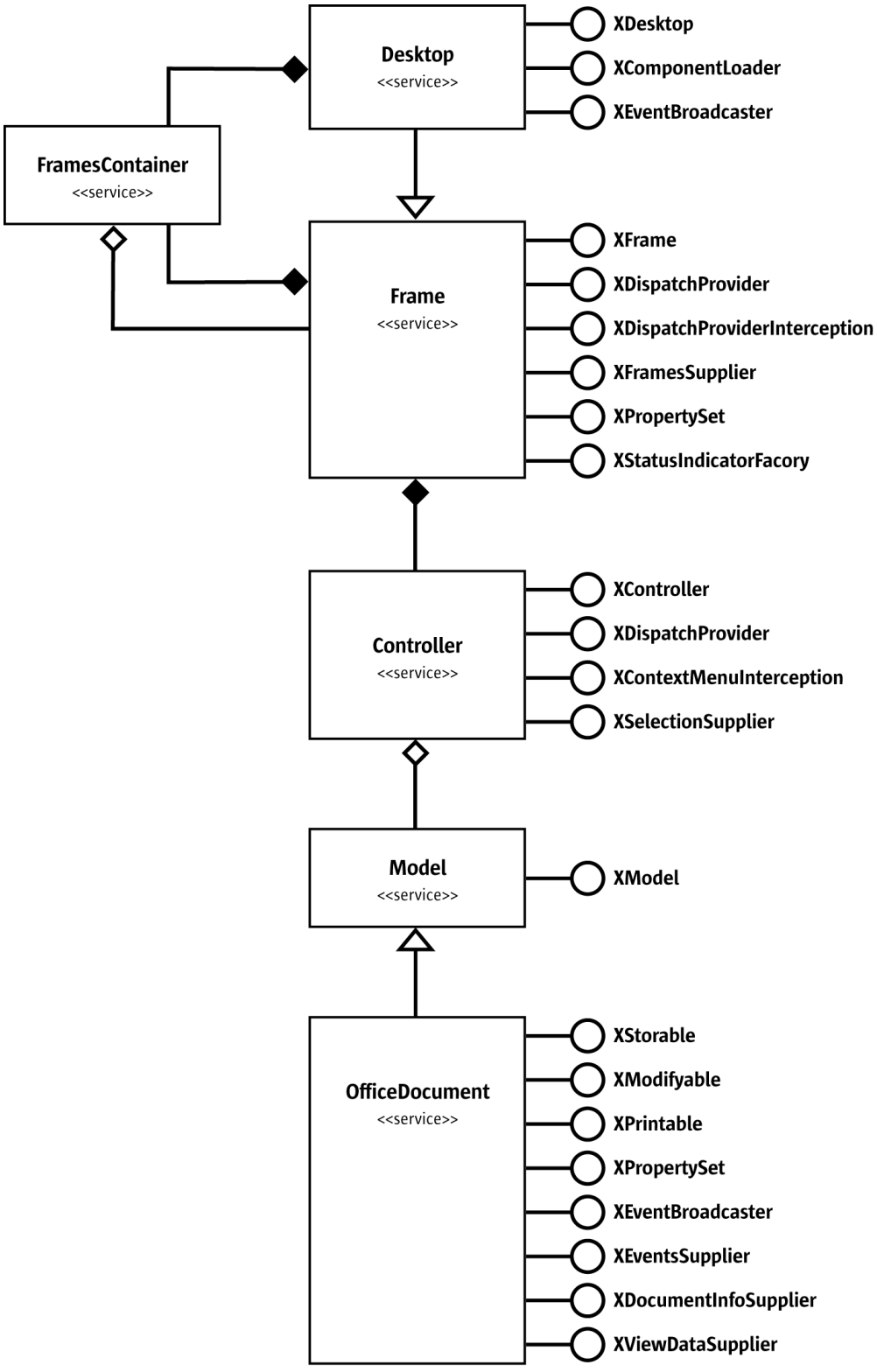


Illustration 54: Desktop Service and Component Framework

The com.sun.star.frame.Desktop service available at the global service manager includes the service com.sun.star.frame.Frame. The Desktop service specification provides three interfaces: com.sun.star.frame.XDesktop, com.sun.star.frame.XComponentLoader and com.sun.star.document.XEventBroadcaster, as shown in the following UML chart:

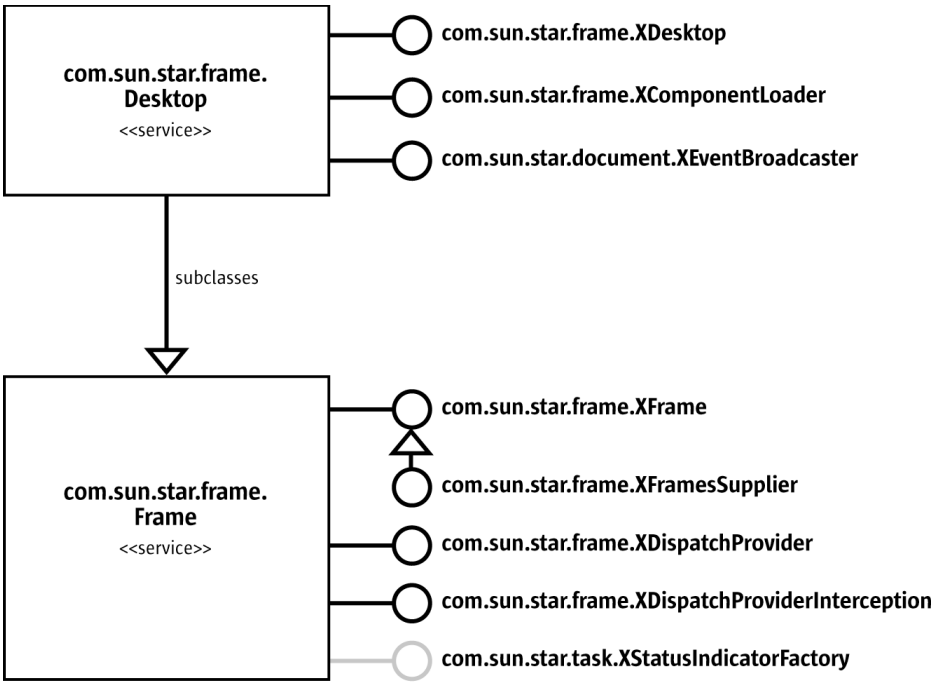


Illustration 55: UML description of the desktop service

The interface com.sun.star.frame.XDesktop provides access to frames and components, and controls the termination of the office process. It defines the following methods:

```
com::sun::star::frame::XFrame getCurrentFrame ()
com::sun::star::container::XEnumerationAccess getComponents ()
com::sun::star::lang::XComponent getCurrentComponent ()
boolean terminate ()
void addTerminateListener ( [in] com::sun::star::frame::XTerminateListener xListener)
void removeTerminateListener ( [in] com::sun::star::frame::XTerminateListener xListener)
```

The methods getCurrentFrame() and getCurrentComponent() distribute the active frame and document model, whereas getComponents() returns a com.sun.star.container.XEnumerationAccess to all loaded documents. For documents loaded in the desktop environment the methods getComponents() and getCurrentComponent() always return the com.sun.star.lang.XComponent interface of the document model.



If a specific document component is required, but are not sure whether this component is the current component, use getComponents() to get an enumeration of all document components, check each for the existence of the com.sun.star.frame.XModel interface and use getURL() at XModel to identify your document. Since not all components have to support XModel, test for XModel before calling getURL().

The office process is usually terminated when the user selects **File - Exit** or after the last application window has been closed. Clients can terminate the office through a call to terminate() and add a terminate listener to veto the shutdown process.

As long as the Windows quickstarter is active, the soffice executable is not terminated.

The following sample shows an com.sun.star.frame.XTerminateListener implementation that prevents the office from being terminated when the class TerminationTest is still active:

```
import com.sun.star.frame.TerminationVetoException;
import com.sun.star.frame.XTerminateListener;
```

```
public class TerminateListener implements XTerminateListener {

    public void notifyTermination (com.sun.star.lang.EventObject eventObject) {
        System.out.println("about to terminate...");
    }

    public void queryTermination (com.sun.star.lang.EventObject eventObject)
        throws TerminationVetoException {

        // test if we can terminate now
        if (TerminationTest.isAtWork() == true) {
            System.out.println("Terminate while we are at work? No way!");
            throw new TerminationVetoException() ; // this will veto the termination,
                                                   // a call to terminate() returns false
        }
    }

    public void disposing (com.sun.star.lang.EventObject eventObject) {
    }
}
```

The following class `TerminationTest` tests the `TerminateListener` above.

```
import com.sun.star.bridge.XUnoUrlResolver;
import com.sun.star.uno.UnoRuntime;
import com.sun.star.uno.XComponentContext;
import com.sun.star.lang.XMultiComponentFactory;
import com.sun.star.beans.XPropertySet;
import com.sun.star.beans.PropertyValue;

import com.sun.star.frame.XDesktop;
import com.sun.star.frame.TerminationVetoException;
import com.sun.star.frame.XTerminateListener;

public class TerminationTest extends java.lang.Object {

    private static boolean atWork = false;

    public static void main(String[] args) {

        XComponentContext xRemoteContext = null;
        XMultiComponentFactory xRemoteServiceManager = null;
        XDesktop xDesktop = null;

        try {
            // connect and retrieve a remote service manager and component context
            XComponentContext xLocalContext =
                com.sun.star.comp.helper.Bootstrap.createInitialComponentContext(null);
            XMultiComponentFactory xLocalServiceManager = xLocalContext.getServiceManager();
            Object urlResolver  = xLocalServiceManager.createInstanceWithContext(
                "com.sun.star.bridge.UnoUrlResolver", xLocalContext );
            XUnoUrlResolver xUnoUrlResolver = (XUnoUrlResolver) UnoRuntime.queryInterface(
                XUnoUrlResolver.class, urlResolver );
            Object initialObject = xUnoUrlResolver.resolve(
                "uno:socket,host=localhost,port=8100;urp;StarOffice.ServiceManager" );
            XPropertySet xPropertySet = (XPropertySet)UnoRuntime.queryInterface(
                XPropertySet.class, initialObject);
            Object context = xPropertySet.getPropertyValue("DefaultContext");
            xRemoteContext = (XComponentContext)UnoRuntime.queryInterface(
                XComponentContext.class, context);
            xRemoteServiceManager = xRemoteContext.getServiceManager();

            // get Desktop instance
            Object desktop = xRemoteServiceManager.createInstanceWithContext (
                "com.sun.star.frame.Desktop ", xRemoteContext);
            xDesktop = (XDesktop)UnoRuntime.queryInterface(XDesktop.class, desktop);

            TerminateListener terminateListener = new TerminateListener ();
            xDesktop.addTerminateListener (terminateListener);

            // try to terminate while we are at work
            atWork = true;
            boolean terminated = xDesktop.terminate ();
            System.out.println("The Office " +
                (terminated == true ? "has been terminated" : "is still running, we are at work"));

            // no longer at work
            atWork = false;
            // once more: try to terminate
            terminated = xDesktop.terminate ();
            System.out.println("The Office " +
                (terminated == true ? "has been terminated" :
                    "is still running. Someone else prevents termination, e.g. the quickstarter"));
        }
        catch (java.lang.Exception e){
```

```
            e.printStackTrace();
        }
        finally {
            System.exit(0);
        }


    }
    public static boolean isAtWork() {
        return atWork;
    }

}
```

The office freezes when terminate() is called if there are unsaved changes. As a workaround set all documents into an unmodified state through their `com.sun.star.util.XModifiable` interface or store them using `com.sun.star.frame.XStorable`.

The Desktop offers a facility to load components through its interface `com.sun.star.frame.XComponentLoader`. It has one method:

```
com::sun::star::lang::XComponent loadComponentFromURL ( [in] string aURL,
                [in] string aTargetFrameName,
                [in] long nSearchFlags,
                [in] sequence < com::sun::star::beans::PropertyValue aArgs > )
```

Refer to chapter *6.1.5 Office Development - OpenOffice.org Application Environment - Handling Documents* for details about the loading process.

For versions beyond 641, the desktop also provides an interface that allows listeners to be notified about certain document events through its interface `com.sun.star.document.XEventBroadcaster`.

```
void addEventListener ( [in] com::sun::star::document::XEventListener xListener)
void removeEventListener ( [in] com::sun::star::document::XEventListener xListener)
```

The XEventListener must implement a single method (besides `disposing()`):

```
[oneway] void notifyEvent ( [in] com::sun::star::document::EventObject Event )
```

The struct `com.sun.star.document.EventObject` has a `string` member `EventName` that assumes one of the values specified in `com.sun.star.document.Events`. The corresponding events are found on the **Events** tab of the **Tools – Configure** dialog when the option OpenOffice.org is selected.

The desktop broadcasts these events for all loaded documents.

The current version of OpenOffice.org does not have a GUI element as a desktop. The redesign of the OpenOffice.org GUI in StarOffice 5.x and later resulted in the `com.sun.star.frame.Frame` service part of the desktop service is now non-functional. While the XFrame interface can still be queried from the desktop, almost all of its methods are dummy implementations. The default implementation of the desktop object in OpenOffice.org is not able to contain a component and refuses to be attached to it, because the desktop is still a frame that is the root for the common hierarchy of all frames in OpenOffice.org. The desktop has to be a frame because its `com.sun.star.frame.XFramesSupplier` interface must be passed to `com.sun.star.frame.XFrame:setCreator()` at the child frames, therefore the desktop becomes the parent frame. However, the following functionality of `com.sun.star.frame.Frame` is still in place:

The desktop interface `com.sun.star.frame.XFramesSupplier` offers methods to access frames. This interface inherits from `com.sun.star.frame.XFrame`, and introduces the following methods:

```
com::sun::star::frame::XFrames getFrames ()
com::sun::star::frame::XFrame getActiveFrame ()
void setActiveFrame ( [in] com::sun::star::frame::XFrame xFrame)
```

The method `getFrames()` returns a `com.sun.star.frame.XFrames` container, that is a `com.sun.star.container.XIndexAccess`, with additional methods to add and remove frames:

```
void append ( [in] com::sun::star::frame::XFrame xFrame )
sequence < com::sun::star::frame::XFrame > queryFrames ( [in] long nSearchFlags )
void remove ( [in] com::sun::star::frame::XFrame xFrame )
```

This `XFrames` collection is used when frames are added to the desktop to become application windows.

Through `getActiveFrame()`, you access the active sub-frame of the desktop frame, whereas `setActiveFrame()` is called by a sub-frame to inform the desktop about the active sub-frame.

The object returned by `getFrames()` does not support `XTypeProvider`, therefore it cannot be used with OpenOffice.org Basic.

The parent interface of `XFramesSupplier`, `com.sun.star.frame.XFrame` is functional by accessing the frame hierarchy below the desktop. These methods are discussed in the section *6.1.3 Office Development - OpenOffice.org Application Environment - Using the Component Framework - Frames* below:

```
com::sun::star::frame::XFrame findFrame ( [in] string aTargetFrameName, [in] long nSearchFlags );
boolean isTop ();
```

The generic dispatch interface `com.sun.star.frame.XDispatchProvider` executes functions of the internal `Desktop` implementation that are not accessible through specialized interfaces. Dispatch functions are described by a command URL. The `XDispatchProvider` returns a dispatch object that dispatches a given command URL. A reference of command URLs supported by the desktop is available on OpenOffice (http://www.openoffice.org/files/documents/25/60/commands_11beta.html). Through the `com.sun.star.frame.XDispatchProviderInterception`, client code intercepts thecommand dispatches at the desktop. The dispatching process is described in section *6.1.6 Office Development - OpenOffice.org Application Environment - Using the Dispatch Framework*.

6.1.3 Using the Component Framework

The component framework comprises the interfaces of frames, controllers and models used to manage components in the OpenOffice.org desktop environment. In our context, everything that "dwells" in a frame of the desktop environment is called a *component,* because the interface `com.sun.star.lang.XComponent` is the common denominator for objects that are loaded into frames.

Frames, controllers and models hold references to each other. The frame is by definition the default owner of the controller and the model, that is,. it is responsible to call `dispose()` on the controller and model when it is destroyed itself. Other objects that are to hold references to the frame, controller, or model must register as listeners to be informed when these references become invalid. Therefore `XModel`, `XController` and `XFrame` inherit from `XComponent`:

```
void dispose ()
void addEventListener ( [in] com::sun::star::lang::XEventListener xListener)
void removeEventListener ( [in] com::sun::star::lang::XEventListener aListener)
```

The process to resolve the circular dependencies of the component framework is a complex. For instance, the objects involved in the process may be in a condition where they may not be disposed of. Refer to the section *6.1.5 Office Development - OpenOffice.org Application Environment - Handling Documents - Closing Documents* for additional details.

Theoretically every UNO object could exist in a frame, as long as it is willing to let the frame control its existence when it ends.

A trivial component (`XWindow` only) is enough for simple viewing purposes, where no activation of a component and related actions like cursor positioning or user interactions are necessary.

If the component participates in more complex interactions, it must implement the controller service.

Many features of the desktop environment are only available if the URL of a component is known. For example:

- Presenting the URL or title of the document.
- Inserting the document into the autosave queue.
- Preventing the desktop environment from loading documents twice.
- Allow for participation in the global document event handling.

In this case, `com.sun.star.frame.XModel` comes into operation, since it has methods to handle URLs, among others.

So a complete office component is made up of

- a controller object that presents the model or shows a view to the model that implements the `com.sun.star.frame.Controller` service, but publishes additional document-specific interfaces. For almost all OpenOffice.org document types there are document specific controller object specifications,such as `com.sun.star.sheet.SpreadsheetView`, and `com.sun.star.drawing.DrawingDocumentDrawView`. For controllers, refer to the section *6.1.3 Office Development - OpenOffice.org Application Environment - Using the Component Framework - Controllers.*
- a model object implementing the `com.sun.star.document.OfficeDocument` service. Refer to the section *6.1.3 Office Development - OpenOffice.org Application Environment - Using the Component Framework - Models.*

Getting Frames, Controllers and Models from Each Other

Usually developers require the controller and frame of an already loaded document model. The `com.sun.star.frame.XModel` interface of OpenOffice.org document models gets the controller that provides access to the frame through its `com.sun.star.frame.XController` interface. The following illustration shows the methods that get the controller and frame for a document model and conversely. From the frame , obtain the corresponding component and container window.

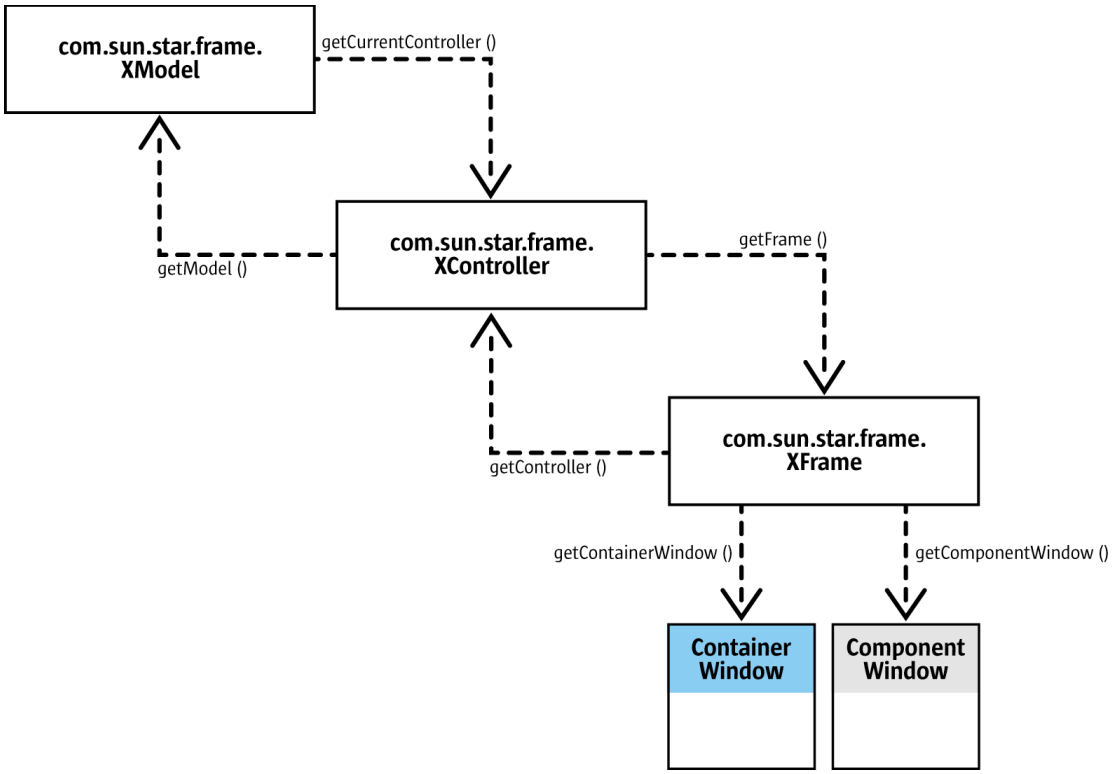


Illustration 56: Frame-Controller-Model Organization

If the loaded component is a trivial component and implements `com.sun.star.awt.XWindow` only, the window and the window peer is reached by querying these interfaces from the `com.sun.star.lang.XComponent` returned by `loadComponentFromURL()`.

Frames

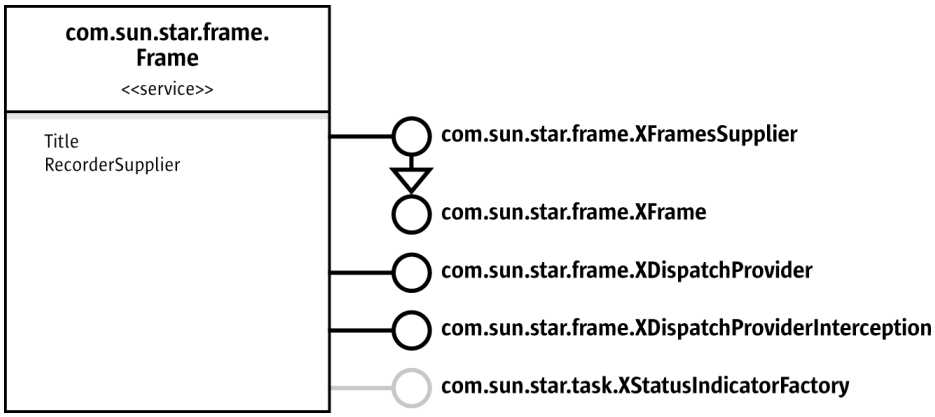


Illustration 57: Frame Service

XFrame

Frame Setup

The main role of a frame is to link components into a surrounding window system. This role is expressed by the following methods of the frame's main interface `com.sun.star.frame.XFrame`:

```
// methods for container window
void initialize ( [in] com::sun::star::awt::XWindow xWindow);
com::sun::star::awt::XWindow getContainerWindow ();

// methods for component window and controller
boolean setComponent ( [in] com::sun::star::awt::XWindow xComponentWindow,
                       [in] com::sun::star::frame::XController xController );
com::sun::star::awt::XWindow getComponentWindow ();
com::sun::star::frame::XController getController ();
```

The first two methods deal with the container window of a frame, the latter three are about linking the component and the component window with the frame. The method `initialize()` expects a top window that is created by the AWT toolkit that becomes the container window of the frame and is retrieved by `getContainerWindow()`.

Frame Hierarchies

When frames link components into a surrounding window system, they build a frame hierarchy. This aspect is covered by the hierarchy-related `XFrame` methods:

```
[oneway] void setCreator ( [in] com::sun::star::frame::XFramesSupplier xCreator );
com::sun::star::frame::XFramesSupplier getCreator ();
string getName ();
[oneway] void setName ( [in] string aName );
com::sun::star::frame::XFrame findFrame ( [in] string aTargetFrameName, [in] long nSearchFlags );
boolean isTop ();
```

The `XFrame` method `setCreator()` informs a frame about its parent frame and must be called by a frames container (`com.sun.star.frame.XFrames`) when a frame is added to it by a call to `com.sun.star.frame.XFrames:append()`. A frames container is provided by frames supporting the interface `com.sun.star.frame.XFramesSupplier`. `XFramesSupplier` is currently supported by the desktop frame and by the default frame implementation used by OpenOffice.org documents. It is described below.

The frame has a custom name that is read through `getName()` and written through `setName()`. Frames in the desktop hierarchy created by GUI interaction usually do not have names. The `getName()` returns an empty string for them, whereas frames that are created for special purposes, such as the beamer frame or the online help, have names. Developers can set a name and use it to address a frame in `findFrame()` calls or when loading a component into the frame. Custom frame names must not start with an underscore.Leading underscores are reserved for special frame names.See below.

Every frame in the frame hierarchy is accessed through any other frame in this hierarchy by calling the `findFrame()` method. This method searches for a frame with a given name in five steps: self, children, siblings, parent, and create if not found. The `findFrame()` checks the called frame, then calls `findFrame()` at its children, then its siblings and at its parent frame. The fifth step in the search strategy is reached if the search makes it to the desktop without finding a frame with the given name. In this case, a new frame is created and assigned the name that was searched for. If the top frame is outside the desktop hierarchy, a new frame is not created.

The name used with `findFrame()` can be an arbitrary string without a leading underscore or one of the following reserved frame names. These names are for internal use for loading documents.Some of the reserved names are logical in a `findFrame()` call, also. A complete list of reserved frame names can be found in section *6.1.5 Office Development - OpenOffice.org Application Environment - Handling Documents - Loading Documents - Target Frame*.

_top
Returns the top frame of the called frame, first frame where `isTop()` returns true when traveling up the hierarchy.

_parent
Returns the next frame above in the frame hierarchy.

_self
Returns the frame itself, same as an empty target frame name. This means you are searching for a frame you already have, but it is legal to do so.

_blank
Creates a new top-level frame whose parent is the desktop frame.

Calls with "`_top`" or "`_parent`" return the frame itself if the called frame is a top frame or has no parent. This is compatible to the targetting strategies of web browsers.

We have seen that `findFrame()` is called recursively. To control the recursion, the search flags parameter specified in the constants group `com.sun.star.frame.FrameSearchFlag` is used. For all of the five steps mentioned above, a suitable flag exists (`SELF, CHILDREN, SIBLINGS, PARENT, CREATE`). Every search step can be prohibited by deleting the appropriate `FrameSearchFlag`. The search flag parameter can also be used to avoid ambiguities caused by multiple occurrences of a frame name in a hierarchy by excluding parts of the frame tree from the search. If `findFrame()` is called for a reserved frame name, the search flags are ignored.



An additional flag can be used to extend a bottom-up search to all OpenOffice.org application windows, no matter where the search starts. Based on the five flags for the five steps, the default frame search stops searching when it reaches a top frame and does not continue with other OpenOffice.org windows. Setting the `TASKS` flag overrides this.

There are separate frame hierarchies that do not interact with each other. If a frame is created, but not inserted into any hierarchy, it becomes the top frame of its own hierarchy. This frame and its contents can not be accessed from other hierarchies by traversing the frame hierarchies through API calls. , Also, this frame and its content cannot reach frames and their contents in other hierarchies. It is the code that creates a frame and decides if the new frame becomes part of an existing hierarchy, thus enabling it to find other frames ,and making it and its viewable component visible to the other frames. Examples for frames that are not inserted into an existing hierarchy are preview frames in dialogs, such as the document preview in the **File – New – Templates and Documents** dialog.



This is the only way the current frame and desktop implementation handle this. If one exchanges either or both of them by another implementation, the treatment of the "`_blank`" target and the `CREATE` SearchFlag may differ.

Frame Actions

Several actions take place at a frame. The context of viewable components can change, a frame may be activated or the relationship between frame and component may be altered. For instance, when the current selection in a document has been changed, the controller informs the frame about it by calling `contextChanged()`. The frame then tells its frame action listeners that the context has changed. The frame action listeners are also informed about changes in the relationship between the frame and component, and about frame activation. The corresponding `XFrame` methods are:

```
void contextChanged ();

[oneway] void activate ();
[oneway] void deactivate ();
boolean isActive ();
```

```
[oneway] void addFrameActionListener ( [in] com::sun::star::frame::XFrameActionListener xListener );
[oneway] void removeFrameActionListener ( [in] com::sun::star::frame::XFrameActionListener xListener
);
```

The method `activate()` makes the given frame the active frame in its parent container. If the parent is the desktop frame, this makes the associated component the current component. However, this is not reflected in the user interface by making the corresponding window the top window. If the container of the active frame is to be the top window, use `setFocus()` at the `com.sun.star.awt.XWindow` interface of the container window.

The interface `com.sun.star.frame.XFrameActionListener` used with `addFrameActionListener()` must implement the following method:

Method of `com.sun.star.frame.XFrameActionListener`	
`frameAction()`	Takes a struct `com.sun.star.frame.FrameActionEvent`. The struct contains two members: the source `com.sun.star.frame.XFrame` `Frame` and an enum `com.sun.star.frame.FrameActionEvent` `Action` value with one of the following values: COMPONENT_ATTACHED: a component has been attached to a frame. This is almost the same as the instantiation of the component within that frame. The component is attached to the frame immediately before this event is broadcast. COMPONENT_DETACHING: a component is detaching from a frame. This is the same as the destruction of the component which was in that frame. The moment the event is broadcast the component is still attached to the frame, but in the next moment it will not be.. COMPONENT_REATTACHED: a component has been attached to a new model. In this case, the component remains the same, but operates on a new model component. FRAME_ACTIVATED: a component has been activated. Activations are broadcast from the top component which was not active, down to the innermost component. FRAME_DEACTIVATING: broadcast immediately before the component is deactivated. Deactivations are broadcast from the innermost component which does not stay active up to the outermost component which does not stay active. CONTEXT_CHANGED: a component has changed its internal context, for example, the selection. If the activation status within a frame changes, this is a context change, also. FRAME_UI_ACTIVATED: broadcast by an active frame when it is getting UI control (tool control). FRAME_UI_DEACTIVATING: broadcast by an active frame when it is losing UI control (tool control).



At this time, the XFrame methods used to build a frame-controller-model relationship can only be fully utilized by frame loader implementations or customized trivial components. Outside a frame loader you can create a frame, but the current implementations cannot create a standalone controller that could be used with setComponent(). Therefore, you can not remove components from one frame and add them to another or create additional controllers for a loaded model using the component framework. This is due to restrictions of the VCL and the C++ implementation of the current document components.

Currently, the only way for clients to construct a frame and insert a OpenOffice.org document into it, is to use the com.sun.star.frame.XComponentLoader interface of the com.sun.star.frame.Desktop or the interfaces com.sun.star.frame.XSynchronousFrameLoader, the preferred frame loader interface, and the asynchronous com.sun.star.frame.XFrameLoader of the com.sun.star.frame.FrameLoader service that is available at the global service factory.

The recommended method to get additional controllers for loaded models is to use the OpenNewView property with loadComponentFromURL() at the com.sun.star.frame.XComponentLoader interface of the desktop.

There is also another possibility: dispatch a ".uno:NewWindow" command to a frame that contains that model.

XFramesSupplier

The Frame interface com.sun.star.frame.XFramesSupplier offers methods to access sub-frames of a frame. The frame implementation of OpenOffice.org supports this interface. This interface inherits from com.sun.star.frame.XFrame, and introduces the following methods:

```
com::sun::star::frame::XFrames getFrames ()
com::sun::star::frame::XFrame getActiveFrame ()
void setActiveFrame ( [in] com::sun::star::frame::XFrame xFrame)
```

The method getFrames() returns a com.sun.star.frame.XFrames container, that is a com.sun.star.container.XIndexAccess with additional methods to add and remove frames:

```
void append ( [in] com::sun::star::frame::XFrame xFrame )
sequence < com::sun::star::frame::XFrame > queryFrames ( [in] long nSearchFlags )
void remove ( [in] com::sun::star::frame::XFrame xFrame );
```

This XFrames collection is used when frames are appended to a frame to become sub-frames. The append() method implementation must extend the existing frame hierarchy by an internal call to setCreator() at the parent frame in the frame hierarchy. The parent frame is always the frame whose XFramesSupplier interface is used to append a new frame.

Through getActiveFrame() access the active sub-frame in a frame with subframes. If there are no sub-frames or a sub-frame is currently non active, the active frame is null. The setActiveFrame() is called by a sub-frame to inform the frame about the activation of the sub-frame. In setActiveFrame(), the method setActiveFrame() at the creator is called, then the registered frame action listeners are notified by an appropriate call to frameAction() with com.sun.star.frame.FrameActionEvent:Action set to FRAME_UI_ACTIVATED.

XDispatchProvider and XDispatchProviderInterception

Frame services also support com.sun.star.frame.XDispatchProvider and com.sun.star.frame.XDispatchProviderInterception. The section *6.1.6 Office Development - OpenOffice.org Application Environment - Using the Dispatch Framework* explains how these interfaces are used.

XStatusIndicatorFactory

The frame implementation supplies a status indicator through its interface com.sun.star.task.XStatusIndicatorFactory. A status indicator can be used by a frame

loader to show the loading process for a document. The factory has only one method that returns an object supporting `com.sun.star.task.XStatusIndicator`:

```
com::sun::star::task::XStatusIndicator createStatusIndicator ()
```

The status indicator is displayed by a call to `com.sun.star.task.XStatusIndicator:start()`. Pass a text and a numeric range, and use `setValue()` to let the status bar grow until the maximum range is reached. The method `end()` removes the status indicator.

Controllers

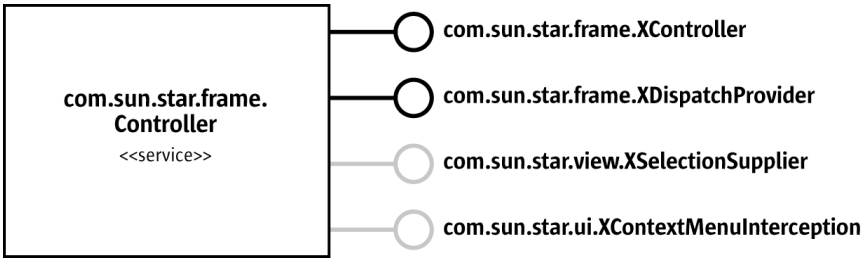


Illustration 58: Controller Service

XController

A `com.sun.star.frame.XController` inherits from `com.sun.star.lang.XComponent` and introduces the following methods:

```
com::sun::star::frame::XFrame getFrame ()
void attachFrame (com::sun::star::frame::XFrame xFrame)
com::sun::star::frame::XModel getModel ()
boolean attachModel (com::sun::star::frame::XModel xModel)
boolean suspend (boolean bSuspend)
any getViewData ()
void restoreViewData (any Data)
```

The `com.sun.star.frame.XController` links model and frame through the methods `get/attachModel()` and `get/attachFrame()`. These methods and the corresponding methods in the `com.sun.star.frame.XModel` and `com.sun.star.frame.XFrame` interfaces act together. calling `attachModel()` at the controller *must* be accompanied by a corresponding call of `connectController()` at the model, and `attachFrame()` at the controller *must* have its counterpart `setComponent()` at the frame.

The controller is asked for permission to dispose of the entire associated component by using `suspend()`. The `suspend()` method shows dialogs, for example, to save changes. To avoid the dialog, close the corresponding frame without using `suspend()` before. The section *6.1.5 Office Development - OpenOffice.org Application Environment - Handling Documents - Closing Documents* provides additional information.

Developers retrieve and restore data used to setup the view at the controller by calling `get/restoreViewData()`. These methods are usually called on loading and saving the document, but they also allow developers to manipulate the state of a view from the outside. The exact content of this data depends on the concrete controller/model pair.

XDispatchProvider

Through `com.sun.star.frame.XDispatchProvider`, the controller participates in the dispatch framework. It is described in section *6.1.6 Office Development - OpenOffice.org Application Environment - Using the Dispatch Framework*.

XSelectionSupplier

The optional `Controller` interface `com.sun.star.view.XSelectionSupplier` accesses the selected object and informs listeners when the selection changes:

```
boolean select ( [in] any aSelection)
any getSelection ()
void addSelectionChangeListener ( [in] com::sun::star::view::XSelectionChangeListener xListener)
void removeSelectionChangeListener ( [in] com::sun::star::view::XSelectionChangeListener xListener)
```

The type of selection depends on the type of the document and the selected object. It is also possible to get the current selection in the active or last controller of a model by calling the method `getCurrentSelection()` in the `com.sun.star.frame.XModel` interface.

XContextMenuInterception

The optional `Controller` interface `com.sun.star.ui.XContextMenuInterception` intercepts requests for context menus in the document's window. See chapter *4.7.5 Writing UNO Components - Integrating Components into OpenOffice.org - Intercepting Context Menus.*

Document Specific Controller Services

The `com.sun.star.frame.Controller` specification is generic and does not describe additional features required for a fully functional document controller specification, such as the controller specifications for Writer, Calc and Draw documents. The following table shows the controller services specified for OpenOffice.org document components.

Once the reference to a controller is retrieved, you can query for these interfaces. Use the `com.sun.star.lang.XServiceInfo` interface of the model to ask it for the supported service(s). The component implementations in OpenOffice.org support the following services. Refer to the related chapters for additional information about the interfaces you get from the controllers of OpenOffice.org documents.

Component and Chapter	Specialized Controller Service	General Description
Writer *7.5 Text Documents - Text Document Controller*	`com.sun.star.text.TextDocumentView`	The text view supplies a text view cursor that has knowledge about the current page layout and page number. It can walk through document pages, screen pages and lines. The selected ruby text is also available, a special Asian text formatting, comparable to superscript.
Calc *8.5 Spreadsheet Documents - Controlling Spreadsheet Documents*	`com.sun.star.sheet.SpreadsheetView`	The spreadsheet view is extremely powerful. It includes the services `com.sun.star.sheet.SpreadsheetViewPane` and `com.sun.star.sheet.SpreadsheetViewSettings`. The view pane handles the currently visible cell range and provides controllers for form controls in the spreadsheet. The view settings deal with the visibility of spreadsheet elements, such as the grid and current zoom mode. Furthermore, the spreadsheet view provides access to the active sheet in the view and the collection of all view panes, allowing to split and freeze the view, and control the interactive selection of a cell range.

Component and Chapter	Specialized Controller Service	General Description
Draw *9.7 Drawing - Drawing and Presentation Document Controller*	`com.sun.star.drawing.DrawingDocumentDrawView`	The drawing document view toggles master page mode and layer mode, controls the current page and supplies the currently visible rectangle.
Impress *9.7 Drawing - Drawing and Presentation Document Controller*	`com.sun.star.drawing.DrawingDocumentDrawView` `com.sun.star.presentation.PresentationView`	The presentation view does not introduce presentation specific features. Running presentations are controlled by the `com.sun.star.presentation.XPresentationSupplier` interface of the presentation document model.
DataBaseAccess	`com.sun.star.sdb.DataSourceBrowser`	This controller has no published functionality that would be useful for developers.
Bibliography	(no special controller specified)	-
Writer (PagePreview)	(no special controller specified)	-
Writer/Webdocument (SourceView)	(no special controller specified)	-
Calc (PagePreview)	(no special controller specified)	-
Chart *10.4 Charts - Chart Document Controller*	(no special controller specified)	-
Math	(no special controller specified)	-

Models

There is not an independent specification for a model service. The interface `com.sun.star.frame.XModel` is currently supported by Writer, Calc, Draw and Impress document components. In our context, we call objects supporting `com.sun.star.frame.XModel`, *model objects*. All OpenOffice.org document components have the service `com.sun.star.document.OfficeDocument` in common. An `OfficeDocument` implements the following interfaces:

XModel

The interface `com.sun.star.frame.XModel` inherits from `com.sun.star.lang.XComponent` and introduces the following methods, which handle the model's resource description, manage its controllers and retrieves the current selection.

```
string getURL ()
sequence < com::sun::star::beans::PropertyValue > getArgs ()
boolean attachResource ( [in] string aURL,
                         [in] sequence < com::sun::star::beans::PropertyValue aArgs > )

com::sun::star::frame::XController getCurrentController ()
void setCurrentController (com::sun::star::frame::XController xController)
void connectController (com::sun::star::frame::XController xController)
void disconnectController (com::sun::star::frame::XController xController)
void lockControllers ()
```

```
void unlockControllers ()
boolean hasControllersLocked ()

com::sun::star::uno::XInterface getCurrentSelection ()
```

The method `getURL()` provides the URL where a document was loaded from or last stored using `storeAsURL()`. As long as a new document has not been saved, the URL is an empty string. The method `getArgs()` returns a sequence of property values that report the resource description according to `com.sun.star.document.MediaDescriptor`, specified on loading or saving with storeAsURL. The method `attachResource()` is used by the frame loader implementations to inform the model about its URL and MediaDescriptor.

The current or last active controller for a model isretrieved through `getCurrentController()`. The corresponding method `setCurrentController()` sets a different current controller at models where additional controllers are available. However, additional controllers can not be created at this time for OpenOffice.org components using the component API. The method `connectController()` is used by frame loader implementations and provides the model with a new controller that has been created for it, without making it the *current* controller. The `disconnectController()` tells the model that a controller may no longer be used. Finally, the model holds back screen updates using `lockControllers()` and `unlockControllers()`. For each call to `lockControllers()`, there must be a call to `unlockControllers()` to remove the lock. The method `hasControllersLocked()` tells if the controllers are locked.

The currently selected object is retrieved by a call to `getCurrentSelection()`. This method is an alternative to `getSelection()` at the `com.sun.star.view.XSelectionSupplier` interface supported by controller services.

XModifiable

The interface `com.sun.star.util.XModifiable` traces the modified status of a document:

```
void addModifyListener ( [in] com::sun::star::util::XModifyListener aListener)
void removeModifyListener ( [in] com::sun::star::util::XModifyListener aListener)
boolean isModified ()
void setModified ( [in] boolean bModified)
```

XStorable

The interface `com.sun.star.frame.XStorable` stores a document under an arbitrary URL or its current location. Details about how to use this interface are discussed in the chapter *6.1.5 Office Development - OpenOffice.org Application Environment - Handling Documents*

XPrintable

The interface `com.sun.star.view.XPrintable` is used to set and get the printer and its settings, and dispatch print jobs. These methods and special printing features for the various document types are described in the chapters *7.2.3 Text Documents - Handling Text Document Files - Printing Text Documents*, *8.2.3 Spreadsheet Documents - Handling Spreadsheet Document Files - Printing Spreadsheet Documents*, *9.2.3 Drawing - Handling Drawing Document Files - Printing Drawing Documents* and *9.4.2 Drawing - Handling Presentation Document Files - Printing Presentation Documents*.

```
sequence< com::sun::star::beans::PropertyValue > getPrinter ()
void setPrinter ( [in] sequence< com::sun::star::beans::PropertyValue > aPrinter )
void print ( [in] sequence< com::sun::star::beans::PropertyValue > xOptions )
```

XEventBroadcaster

For versions later than 641, the optional interface `com.sun.star.document.XEventBroadcaster` at office documents enables developers to add listeners for events related to office documents in

general, or for events specific for the individual document type.See *6.2.8 Office Development - Common Application Features - Document Events*).

```
void addEventListener ( [in] com::sun::star::document::XEventListener xListener)
void removeEventListener ( [in] com::sun::star::document::XEventListener xListener)
```

The XEventListener must implement a single method, besides `disposing()`:

```
[oneway] void notifyEvent ( [in] com::sun::star::document::EventObject Event )
```

The struct `com.sun.star.document.EventObject` has a string member `EventName`, that assumes one of the values specified in `com.sun.star.document.Events`. These events are also on the **Events** tab of the **Tools – Configure** dialog.

The general events are the same events as those provided at the `XEventBroadcaster` interface of the desktop. While the model is only concerned about its own events, the desktop broadcasts the events for all the loaded documents.

XEventsSupplier

The optional interface `com.sun.star.document.XEventsSupplier` binds the execution of dispatch URLs to document events, thus providing a configurable event listener as a simplification for the more general event broadcaster or listener mechanism of the `com.sun.star.document.XEventBroadcaster` interface. This is done programmatically versus manually in **Tools – Configure – Events**.

XDocumentInfoSupplier

The optional interface `com.sun.star.document.XDocumentInfoSupplier` provides access to document information as described in section *6.2.7 Office Development - Common Application Features - Document Info.*Document information is presented in the **File – Properties** dialog in the GUI.

XViewDataSupplier

The optional `com.sun.star.document.XViewDataSupplier` interface sets and restores view data.

```
com::sun::star::container::XIndexAccess getViewData ()
void setViewData ( [in] com::sun::star::container::XIndexAccess aData)
```

The view data are a `com.sun.star.container.XIndexAccess` to sequences of `com.sun.star.beans.PropertyValue` structs. Each sequence represents the settings of a view to the model that supplies the view data.

Document Specific Features

Every service specification for real model objects provides more interfaces that constitute the actual model functionality For example, a text document service `com.sun.star.text.TextDocument` provides text related interfaces. Having received a reference to a model, developers query for these interfaces. The `com.sun.star.lang.XServiceInfo` interface of a model can be used to ask for supported services. The OpenOffice.org document types support the following services:

Document	Service	Chapter
Calc	`com.sun.star.sheet.SpreadsheetDocument`	*8 Spreadsheet Documents*
Draw	`com.sun.star.drawing.DrawingDocument`	*9 Drawing*

Document	Service	Chapter
Impress	com.sun.star.presentation.PresentationDocument	*9 Drawing*
Math	com.sun.star.formula.FormulaProperties	-
Writer (all Writer modules)	com.sun.star.text.TextDocument	*7 Text Documents*
Chart	com.sun.star.chart.ChartDocument	*10 Charts*

Refer to the related chapters for additional information about the interfaces of the documents of OpenOffice.org.

Window Interfaces

The window interfaces of the component window and container window control the OpenOffice.org application windows. This chapter provides a short overview.

XWindow

The interface com.sun.star.awt.XWindow is supported by the component and controller windows. This interface comprises methods to resize a window, control its visibility, enable and disable it, and make it the focus for input device events. Listeners are informed about window events.

```
[oneway] void setPosSize ( long X, long Y, long Width, long Height, short Flags );
com::sun::star::awt::Rectangle getPosSize ();

[oneway] void setVisible ( boolean Visible );
[oneway] void setEnable ( boolean Enable );
[oneway] void setFocus ();

[oneway] void addWindowListener ( com::sun::star::awt::XWindowListener xListener );
[oneway] void removeWindowListener ( com::sun::star::awt::XWindowListener xListener );
[oneway] void addFocusListener ( com::sun::star::awt::XFocusListener xListener );
[oneway] void removeFocusListener ( com::sun::star::awt::XFocusListener xListener );
[oneway] void addKeyListener ( com::sun::star::awt::XKeyListener xListener );
[oneway] void removeKeyListener ( com::sun::star::awt::XKeyListener xListener );
[oneway] void addMouseListener ( com::sun::star::awt::XMouseListener xListener );
[oneway] void removeMouseListener ( com::sun::star::awt::XMouseListener xListener );
[oneway] void addMouseMotionListener ( com::sun::star::awt::XMouseMotionListener xListener );
[oneway] void removeMouseMotionListener ( com::sun::star::awt::XMouseMotionListener xListener );
[oneway] void addPaintListener ( com::sun::star::awt::XPaintListener xListener );
[oneway] void removePaintListener ( com::sun::star::awt::XPaintListener xListener );
```

The com.sun.star.awt.XWindowListener gets the following notifications. The com.sun.star.awt.WindowEvent has members describing the size and position of the window.

```
[oneway] void windowResized ( [in] com::sun::star::awt::WindowEvent e )
[oneway] void windowMoved ( [in] com::sun::star::awt::WindowEvent e )
[oneway] void windowShown ( [in] com::sun::star::lang::EventObject e )
[oneway] void windowHidden ( [in] com::sun::star::lang::EventObject e );
```

What the other listeners do are evident by their names.

XTopWindow

The interface com.sun.star.awt.XTopWindow is available at container windows. It informs listeners about top window events, and it can put itself in front of other windows or withdraw into the background. It also has a method to control the current menu bar:

```
[oneway] void addTopWindowListener ( com::sun::star::awt::XTopWindowListener xListener );
[oneway] void removeTopWindowListener ( com::sun::star::awt::XTopWindowListener xListener );
```

```
[oneway] void toFront ();
[oneway] void toBack ();
[oneway] void setMenuBar ( com::sun::star::awt::XMenuBar xMenu );
```



Although the XTopWindow interface has a method `setMenuBar()`, this method is not usable at this time. The `com.sun.star.awt.XMenuBar` interface is deprecated.

The top window listener receives the following messages. All methods take a `com.sun.star.awt.WindowEvent` with members describing the size and position of the window.

```
[oneway] void windowOpened ( [in] com::sun::star::awt::WindowEvent e )
[oneway] void windowClosing ( [in] com::sun::star::awt::WindowEvent e )
[oneway] void windowClosed ( [in] com::sun::star::awt::WindowEvent e )
[oneway] void windowMinimized ( [in] com::sun::star::awt::WindowEvent e )
[oneway] void windowNormalized ( [in] com::sun::star::awt::WindowEvent e )
[oneway] void windowActivated ( [in] com::sun::star::awt::WindowEvent e )
[oneway] void windowDeactivated ( [in] com::sun::star::awt::WindowEvent e )
```

XWindowPeer

Each XWindow has a `com.sun.star.awt.XWindowPeer`. The `com.sun.star.awt.XWindowPeer` interface accesses the window toolkit implementation used to create it and provides the pointer of the pointing device, and controls the background color. It is also used to invalidate a window or portions of it to trigger a redraw cycle.

```
com::sun::star::awt::XToolkit getToolkit ()
[oneway] void setPointer ( [in] com::sun::star::awt::XPointer Pointer )
[oneway] void setBackground ( [in] long Color )
[oneway] void invalidate ( [in] short Flags )
[oneway] void invalidateRect ( [in] com::sun::star::awt::Rectangle Rect,
                               [in] short Flags )
```

6.1.4 Creating Frames Manually

Frame Creation

]Every time a frame is needed in OpenOffice.org, the `com.sun.star.frame.Frame` service is created. OpenOffice.org has an implementation for this service, available at the global service manager.

This service can be replaced by a different implementation, for example, your own implementation in Java, by registering it at the service manager. In special cases, it is possible to use a custom frame implementation instead of the `com.sun.star.frame.Frame` service by instantiating a specific implementation using the implementation name with the factory methods of the service manager. Both methods can alter the default window and document handling in OpenOffice.org, thus changing or extending its functionality.

Assigning Windows to Frames

Every frame can be assigned to any OpenOffice.org window. For instance, the same frame implementation is used to load a component into an application window of the underlying windowing system or into a preview window of a OpenOffice.org dialog. The `com.sun.star.frame.Frame` service implementation does not depend on the type of the window, although the entirety of the frame and window will be a different object by the user.

If you have a window in your application and want to load a OpenOffice.org document, create a frame and window object, and put them together by a call to `initialize()`. A default frame is created by instantiating an object implementing the `com.sun.star.frame.Frame` service at the global service manager. For window creation, the current `com.sun.star.awt` implementation has

to be used to create windows in all languages supporting UNO. This toolkit offers a method to create window objects that wrap a platform specific window, such as a Java AWT window or a Windows system window represented by its window handle. A Java example is given below.

Two conditions apply to windows that are to be used with OpenOffice.org frames.

The first condition is that the window must be created by the current `com.sun.star.awt.Toolkit` service implementation. Not every object implementing the `com.sun.star.awt.XWindow` interface is used as an argument in the `initialize()` method, because it is syntactically correct, but it is restricted to objects created by the current `com.sun.star.awt` implementation. The insertion of a component into a frame only works if *all* involved windows are .xbl created by the same toolkit implementation. All internal office components, such as Writer and Calc, are implemented using the Visual Component Library (VCL), so that they do not work if the container window is not implemented by VCL. The current toolkit uses this library internally, so all the windows created by the awt toolkit are passed to a frame.No others work at this time. Using VCL directly is not recommended. The code has to be rewritten, whenever this complication has incurred by the current office implementation and is removed, and the toolkit implementation is exchangeable.

The second condition is that if a frame and its component are supposed to get `windowActivated()` messages, the window object implements the additional interface `com.sun.star.awt.XTopWindow`. This is necessary for editing components, because the `windowActivated` event shows a cursor or a selection in the document. As long as this condition is met, further code is not necessary for the interaction between the frame and window, because the frame gets all the necessary events from the window by registering the appropriate listeners in the call to `initialize()`.

When you use the `com.sun.star.awt.Toolkit` to create windows, supply a `com.sun.star.awt.WindowDescriptor` struct to describe what kind of window is required. Set the `Type` member of this struct to `com.sun.star.awt.WindowClass:TOP` and the `WindowService-Name` member to `"window"` if you want to have an application window, or to `"dockingwindow"` if a window is need to be inserted in other windows created by the toolkit.

Setting Components into Frame Hierarchies

Once a frame has been initialized with a window, it can be added to a frames supplier, such as the desktop using the frames container provided by `com.sun.star.frame.XFramesSupplier:getFrames()`. Its method `com.sun.star.frame.XFrames:append()` inserts the new frame into the `XFrames` container and calls `setCreator()` at the new frame, passing the `XFramesSupplier` interface of the parent frame.



The parent frame *must* be set as the creator of the newly created frame. The current implementation of the frames container calls `setCreator()` internally when frames are added to it using `append()`.

The following example creates a new window and a frame, plugs them together, and adds them to the desktop, thus creating a new, empty OpenOffice.org application window. (OfficeDev/DesktopEnvironment/FunctionHelper.java)

```
// Conditions: xSMGR = m_xServiceManager
// Get access to vcl toolkit of remote office to create
// the container window of new target frame.
com.sun.star.awt.XToolkit xToolkit =
 (com.sun.star.awt.XToolkit)UnoRuntime.queryInterface(
  com.sun.star.awt.XToolkit.class,
  xSMGR.createInstance("com.sun.star.awt.Toolkit") );

// Describe the properties of the container window.
// Tip: It is possible to use native window handle of a java window
// as parent for this. see chapter "OfficeBean" for further informations
com.sun.star.awt.WindowDescriptor aDescriptor =
 new com.sun.star.awt.WindowDescriptor();
```

```
 aDescriptor.Type              = com.sun.star.awt.WindowClass.TOP ;
 aDescriptor.WindowServiceName = "window" ;
 aDescriptor.ParentIndex       = -1;
 aDescriptor.Parent            = null;
 aDescriptor.Bounds            = new com.sun.star.awt.Rectangle(0,0,0,0);
 aDescriptor.WindowAttributes  =
  com.sun.star.awt.WindowAttribute.BORDER    |
  com.sun.star.awt.WindowAttribute.MOVEABLE  |
  com.sun.star.awt.WindowAttribute.SIZEABLE  |
  com.sun.star.awt.WindowAttribute.CLOSEABLE ;

 com.sun.star.awt.XWindowPeer xPeer = xToolkit.createWindow(aDescriptor) ;

 com.sun.star.awt.XWindow xWindow = (com.sun.star.awt.XWindow)UnoRuntime.queryInterface (
   com.sun.star.awt.XWindow .class, xPeer);

 // Create a new empty target frame.
 // Attention: Before OpenOffice.org build 643 we must use
 // com.sun.star.frame.Task instead of com.sun.star.frame.Frame,
// because the desktop environment accepts only this special frame type
 // as direct children. It will be deprecated from build 643
 xFrame = (com.sun.star.frame.XFrame)UnoRuntime.queryInterface(
    com.sun.star.frame.XFrame.class,
        xSMGR.createInstance ("com.sun.star.frame.Task "));

 // Set the container window on it.
 xFrame.initialize(xWindow) ;

// Insert the new frame in desktop hierarchy.
 // Use XFrames interface to do so. It provides access to the
 // child frame container of the parent node.
 // Note: append(xFrame) calls xFrame.setCreator(Desktop) automaticly.
 com.sun.star.frame.XFramesSupplier xTreeRoot =
 (com.sun.star.frame.XFramesSupplier)UnoRuntime.queryInterface(
        com.sun.star.frame.XFramesSupplier.class,
            xSMGR.createInstance("com.sun.star.frame.Desktop") );

com.sun.star.frame.XFrames xChildContainer = xTreeRoot.getFrames ();
xChildContainer.append(xFrame) ;

// Make some other initializations.
 xPeer.setBackground(0xFFFFFFFF);
 xWindow.setVisible(true);
 xFrame.setName("newly created 1") ;
```

6.1.5 Handling Documents

Loading Documents

The framework API defines a simple but powerful interface to load viewable components, the `com.sun.star.frame.XComponentLoader`. This interface is implemented by the globally accessible `com.sun.star.frame.Desktop` service,to query the `XComponentLoader` from the desktop.

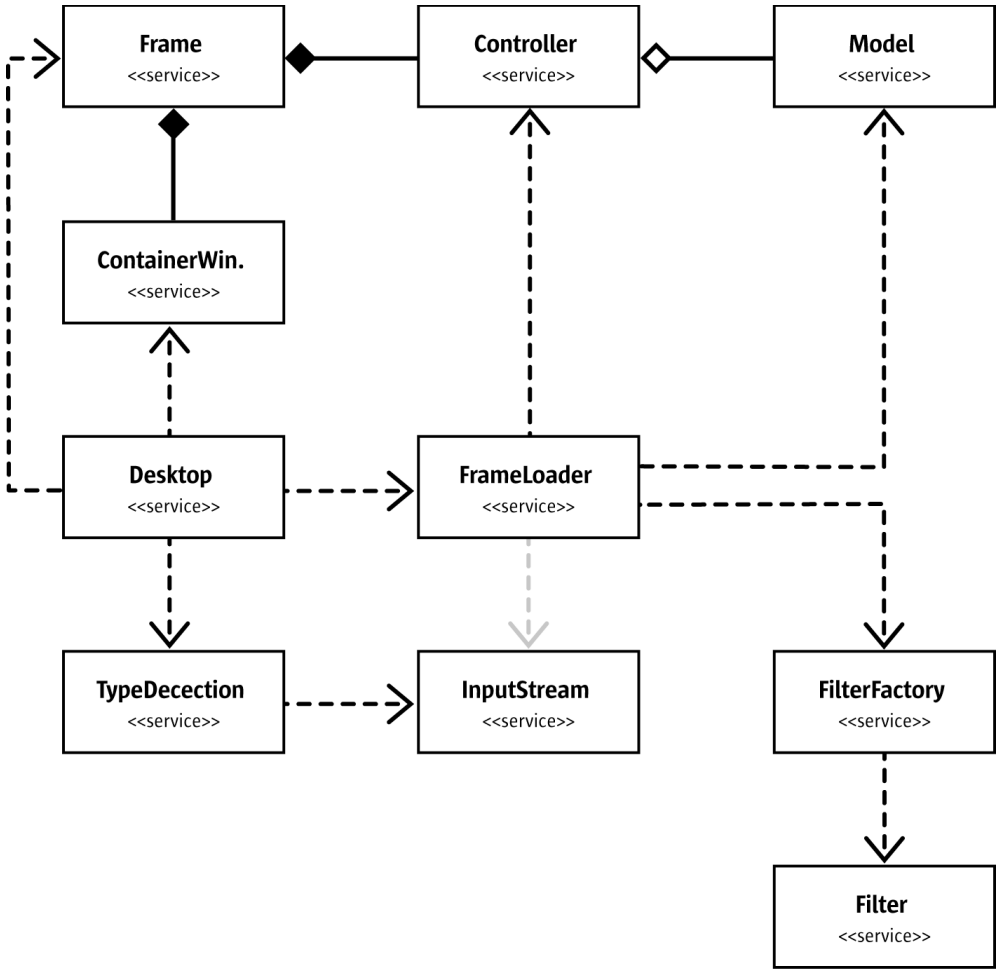


Illustration 59: Services Involved in Document Loading

The interface `com.sun.star.frame.XComponentLoader` has one method:

```
com::sun::star::lang::XComponent loadComponentFromURL ( [in] string aURL,
                [in] string aTargetFrameName,
                [in] long nSearchFlags,
                [in] sequence < com::sun::star::beans::PropertyValue aArgs > )
```

The use fo this method is demonstrated below in the service `com.sun.star.document.MediaDescriptor`.

MediaDescriptor

A call to `loadComponentFromURL()` receives a sequence of `com.sun.star.beans.PropertyValue` structs as a parameter, which implements the `com.sun.star.document.MediaDescriptor` service, consisting of property definitions. It describes where a resource or *medium* should be loaded from and how this should be done.

The media descriptor is also used for saving a document to a location using the interface `com.sun.star.frame.XStorable`. It transports the "where to" and the "how" of the storing procedure. The table below shows the properties defined in the media descriptor.

Some properties are used for loading and saving while others apply to one or the other. If a media descriptor is used, only a few of the members are specified. The others assume default values. Strings default to empty strings in general and interface references default to empty references. For all other properties, the default values are specified in the description column of the table.

Some properties are tagged deprecated. There are old implementations that still use these properties. They are supported, but are discouraged to use them. Use the new property that can be found in the description column of the deprecated property.

To develop a UNO component that uses the media descriptor, note that all the properties are under control of the framework API. Never create your own property names for the media descriptor, or name clashes may be induced if the framework defines a property that uses the same name. Instead, use the `ComponentData` property to transport document specific information. `ComponentData` is specified to be an `any`, therefore it can be a sequence of property values by itself. If you do use it. make an appropriate specification available to users of your component.

Properties of `com.sun.star.document.MediaDescriptor`	
`AsTemplate`	boolean. Setting `AsTemplate` to true creates a new untitled document out of the loaded document, even if it has no template extension. Loading a template, that is, a document with a template extension, creates a new untitled document by default, but setting the `AsTemplate` property to false loads a template for editing.
`Author`	string. Only for storing versions in components supporting versioning: author of version.
`CharacterSet`	string. Defines the character set for document formats that contain single byte characters, if necessary. Which character set names are valid depends on the filter implementation, but with the current filters you can employ the character sets used for the conversion of byte to unicode strings.
`Comment`	string. Only for storing versions in components supporting versioning: comment (description) for stored version.
`ComponentData`	any. This is a parameter that is used for any properties specific for a special office component type.
`FileName` - deprecated	string. Same as URL (added for compatibility reasons)
`FilterData`	any. This is a parameter that is used for any properties specific for a special filter type.
`FilterName`	string. Name of a filter that should be used for loading or storing the component. Names must match the names of the typedetection configuration.Invalid names are ignored. If a name is specified on loading, it will be verified by a filter detection, but in case of doubt it will be preferred.
`FilterFlags` - deprecated	string. For compatibility reasons: same as FilterOptions
`FilterOptions`	string. Some filters need additional parameters. Use only together with property FilterName. Details must be documented by the filter. This is an old format for some filters. If a string is not enough, filters can use the property FilterData.
`Hidden`	boolean. Defines if the loaded component is made visible. If this property is not specified, the component is made visible by default. Making a hidden component visible by calling `setVisible()` at the container window is not recommended at this time.

Properties of `com.sun.star.document.MediaDescriptor`	
`InputStream`	com.sun.star.io.XInputStream. Used when loading a document. Reading must be done using this stream. If no stream is provided, the loader creates a stream by itself using the URL, version number, readonly flag, password, or anything requiredfor stream creation, given in the media descriptor. The model becomes the final owner of the stream and usually holds the reference to lock the file. Therefore, it is not allowed to keep a reference to this `InputStream` after loading the component.It is useless, because an InputStream is only usable once for reading. Even if it implements the `com.sun.star.io.XSeekable` interface, do not interfere with the model's reading process. Consider all the objects involved in the loading process as temporary.
`InteractionHandler`	com.sun.star.task.XInteractionHandler. Object implementing the `com.sun.star.task.InteractionHandler` service that handles exceptional situations where proceeding with the task is impossible without additional information or impossible at all. OpenOffice.org provides a default implementation that can handle many situations. If no InteractionHandler is set, a suitable exception is thrown. It is not allowed to keep a reference to this object, not even in the loaded or stored components' copy of the MediaDescriptor provided by its arguments attribute.
`JumpMark`	string. Jump to a marked position after loading. The office document loaders expect simple strings used like targets in HTML documents.Do not use a leading # character. The meaning of a jump mark depends upon the filter, but in Writer, bookmarks can be used, whereas in Calc cells, cell ranges and named areas are supported.
`MediaType` (string)	string. Type of the medium to load that must match to one of the types defined in the typedetection configuration, otherwise it is ignored. The typedetection configuration is found in the *TypeDetection.xml* file in the *config/registry/instance/org/openoffice/Office* folders of the user or share tree. The MediaType is found in the "type" entries. Here, it is the second member of the "data" value. This parameter bypasses the type detection of the desktop environment, so that passing a wrong MediaType causes load failures.
`OpenFlags` - deprecated	string. For compatibility reasons: string that summarizes some flags for loading. The string contains capital letters for the flags: "ReadOnly" - "R" "Preview" - "B" "AsTemplate" - "T" "Hidden" - "H" Use the corresponding boolean parameters instead.
`OpenNewView`	boolean. Affects the behavior of the component loader when a resource is already loaded. If true, the loader tries to open a new view for a document already loaded. For components supporting multiple views, a second window is opened as if the user clicked **Window – New Window.**Other components are loaded one more time. Without this property, the default behavior of the loader applies, for example, the loader of the desktop activates a document if the user tries to load it a second time.

Properties of `com.sun.star.document.MediaDescriptor`	
`Overwrite`	boolean. For storing only: overwrite existing files with the same name, default is true, so an `com.sun.star.io.IOException` occurs if the target file already exists. If the default is changed and the file exists, the UCB throws an exception. If the file is loaded through API, this exception is transported to the caller or handled by an interaction handler.
`Password`	string. A password for loading or storing a component, if necessary. If no password is specified, loading of a password protected document fails, storing is done without encryption.
`PostData`	reference <XinputStream>. HTTP post data to send to a location described by the media descriptor to get a result that is loaded as a component, usually in webforms. Default is: no PostData.
`PostString` - deprecated	string. Same as PostData, but the data is transferred as a string (just for compatibility).
`Preview`	boolean. Setting this to true tells the loaded component that it is loaded as a preview, so that it can optimize loading and viewing for this special purpose. Default is false.
`ReadOnly`	boolean. Tells if a document is to be loaded in a (logical) readonly or in read/write mode. If opening in the desired mode is impossible, an error occurs. By default, the loaded content decides what to do. If its UCB content supports a "readonly" property, the logical open mode depends on that property, otherwise it is read/write. This property only affects the UI. Opening a document in read only mode does not prevent the component from being modified by API calls, but all modifying functionality in the UI is disabled or removed.
`Referer` (the wrong spelling is kept for compatibility reasons)	string. A URL describing the environment of the request; for example,. a referrer may be the URL of a document, if a hyperlink inside this document is clicked to load another document. The referrer may be evaluated by the addressed UCB content or the loaded document. Without a referrer, the processing of URLs that require security checks is denied, for instance `macro:` URLs.
`StatusIndicator`	`com.sun.star.task.XStatusIndicator`. Object implementing the `com.sun.star.task.XStatusIndicator` interface that gives status information, such as text or progress, for the target frame. OpenOffice.org provides a default implementation that is retrieved by calling `createStatusIndicator()` at the frame you load a component into. Usually you do not need this parameter if you do not want to use any other indicator than the one in the status bar of the document window. It is not allowed to keep a reference to this object, not even in the loaded or stored component's copy of the MediaDescriptor provided by its `getArgs()` method.
`TemplateName`	string. The logical name of a template to load. Together with the TemplateRegionName property this is used instead of the URL of the template. The logical names are the template names you see in the templates dialog.
`TemplateRegionName`	string. See TemplateName. The template region names are the folder names you see in the templates dialog.

Properties of com.sun.star.document.MediaDescriptor	
Unpacked	boolean. For storing: Setting this to true means that a zip file is not used to save the document. Use a folder instead for UCB contents that support folders, such as file, WebDAV, and ftp. Default is false.
URL	string. The location of the component in URL syntax.
Version	short. For components supporting versioning: the number of the version to be loaded or saved. Default is zero and means that no version is created or loaded, and the main document is processed.
ViewData	any. Data to set a special view state after loading. The type depends on the component and is retrieved from a controller object by its com.sun.star.document.XViewDataSupplier interface. Default is: no ViewData.
ViewId	short. For components supporting different views: a number to define the view that should be constructed after loading. Default is: zero, and this should be treated by the component as the default view.
MacroExecutionMode	short. How should the macro be executed - the value should be one from com.sun.star.document.MacroExecMode constants group
UpdateDocMode	short. Can the document be updated depending on links. The value should be one from com.sun.star.document.UpdateDocMode constant group

The media descriptor used for loading and storing components is passed as an in/out parameter to some objects that participate in the loading or storing process, that is, the com.sun.star.document.TypeDetection service or a com.sun.star.document.ExtendedTypeDetection service. These objects add additional information they have gathered to the media descriptor, so that other objects called later do not have to reinvestigate this.

The first object that gets the media descriptor might need an input stream, but assume that there is currently none. The object creates one and uses it. If the stream happens to be seekable (usually it is), the object puts the stream into the media descriptor, so that it passes it to other objects that need the stream as well. They do not have to create it again. It is important for streams created for a remote resource, such as http contents.

If the stream, provided from the outside or created by the first consumer, is not seekable, every consumer creates one. It creates a buffering stream component that reads in the original stream and provides a seekable stream for all further consumers. This buffered stream can be put into the media descriptor.

As previously mentioned, the easiest way to load a document is to call loadComponentFromURL() at the desktop service, but any other object could implement this interface.

URL Parameter

The URL is part of the media descriptor and also an explicit parameter for loadComponentFromURL(). This enables script code to load a document without creating a media descriptor at the cost of code redundancy. The URL parameter of loadComponentFromURL() overrides a possible URL property passed in the media descriptor. Aside from valid URLs that describe an existing file, the following URLs are used to open viewable components in OpenOffice.org:

Component	URL
Writer	private:factory/swriter
Calc	private:factory/scalc

Component	URL
Draw	private:factory/sdraw
Impress	private:factory/simpress
Database	.component:DB/QueryDesign .component:DB/TableDesign .component:DB/RelationDesign .component:DB/DataSourceBrowser .component:DB/FormGridView
Bibliography	.component:Bibliography/View1

Target Frame

The URL and media descriptor `loadComponentFromURL()` have two additional arguments, the target frame name and search flags. The method `loadComponentFromURL()` looks for a frame in the frame hierarchy and loads the component into the frame it finds. It uses the same algorithm as `findFrame()` at the `com.sun.star.frame.XFrame` interface, described in section *6.1.3 Office Development - OpenOffice.org Application Environment - Using the Component Framework - Frames - XFrame - Frame Hierarchies*.

The target frame name is a reserved name starting with an underscore or arbitrary name. The reserved names denote frequently used frames in the frame hierarchy or special functions, whereas an arbitrary name is searched recursively. If a reserved name is used, the search flags are ignored and set to `0`. The following reserved names are supported:

_self
: Returns the frame itself. The same as with an empty target frame name. This means to search for a frame you already have, but it is legal.

_top
: Returns the top frame of the called frame .,The first frame where `isTop()` returns true when traveling up the hierarchy. If the starting frame does not have a parent frame, the call is treated as a search for `"_self"`. This behavior is compatible to the frame targeting in a web browser.

_parent
: Returns the next frame above in the frame hierarchy. If the starting frame does not have a parent frame, the call is treated as a search for `"_self"`. This behavior is compatible to the frame targeting in a web browser.

_blank
: Creates a new top-level frame as a child frame of the desktop. If the called frame is not part of the desktop hierarchy, this call fails. Using the `"_blank"` target loads open documents again that result in a read-only document, depending on the UCB content provider for the component. If loading is done as a result of a user action, this becomes confusing to theusers, therefore the `"_default"` target is recommended in calls from a user interface, instead of `"_blank"`. Refer to the next section for a discussion about the _default target..

_default
: Similar to `"_blank"`, but the implementation defines further behavior that has to be documented by the implementer. The `com.sun.star.frame.XComponentLoader` implemented at the desktop object shows the following default behavior.

  First, it checks if the component to load is already loaded in another top-level frame. If this is the case, the frame is activated and brought to the foreground. When the `OpenNewView` property is set to true in the media descriptor, the loader creates a second controller to show another view for the loaded document. For components supporting this, a second window is opened as

if the user clicked **Window – New Window.** The other components are loaded one more time, as if the "_blank" target had been used. Currently, almost all office components implementing com.sun.star.frame.XModel have multiple controllers, except for HTML and writer documents in the online view. The database and bibliography components have no model, therefore they cannot open a second view at all and OpenNewView leads to an exception with them.

Next, the loader checks if the active frame contains an unmodified, empty document of the same document type as the component that is being loaded. If so, the component is loaded into that frame, replacing the empty document, otherwise a new top-level frame is created similar to a call with "_blank".

Names starting with an underscore must not be used as real names for a frame.

If the given frame name is an arbitrary string, the loader searches for this frame in the frame hierarchy. The search is done in the following order: self, children, siblings, parent, create if not found. Each of these search steps can be skipped by deleting it from the com.sun.star.frame.FrameSearchFlag bit vector:

Constants in com.sun.star.frame.FrameSearchFlag group	
SELF	search current frame
CHILDREN	search children recursively
SIBLINGS	search frames on the same level
PARENT	search frame above the current frame in the hierarchy
CREATE	create new frame if not found
TASKS	do not stop searching when a top frame is reached, but continue with other top frames
ALL	search the frame hierarchy below the current top frame, do not create new frame: SELF | CHILDREN | SIBLINGS | PARENT
GLOBAL	search all frames, do not create new frame: SELF | CHILDREN | SIBLINGS | PARENT | TASKS

A typical case for a named frame is a situation where a frame is needed to be reused for subsequent loading of components, for example, a frame attached to a preview window or a docked frame, such as the frame in OpenOffice.org that opens the address book when the **F4** key is pressed.

The frame names "_self", "_top" and "_parent" define a frame target relative to a starting frame. They can only be used if the component loader interface finds the frame and the setComponent() can be used with the frame. The desktop frame is the root, therefore it does not have a top and parent frame. The component loader of the desktop cannot use these names, because the desktop refuses to have a component set into it.. However, if a frame implemented com.sun.star.frame.XComponentLoader, these names could be used.



OpenOffice.org 1.1 will have a frame implementation that supports XComponentLoader.

The reserved frame names are also used as a targeting mechanism in the dispatch framework with regard to as far as the relative frame names being resolved. For additional information, see chapter *6.1.6 Office Development - OpenOffice.org Application Environment - Using the Dispatch Framework*

The example below creates a frame, and uses the target frame and search flag parameters of loadComponentFromURL() to load a document into it.
(OfficeDev/DesktopEnvironment/FunctionHelper.java)

```
// Conditions: sURL = "private:factory/swriter"
//             xSMGR = m_xServiceManager
//             xFrame = reference to a frame
```

```
//            lProperties[] = new com.sun.star.beans.PropertyValue[0]

// First prepare frame for loading.
// We must adress it inside the frame tree without any complications.
// To do so we set an unambiguous name and use it later.
// Don't forget to reset the name to the original name after that.

String sOldName = xFrame.getName();
String sTarget  = "odk_officedev_desk";
xFrame.setName(sTarget);

// Get access to the global component loader of the office
// for synchronous loading of documents.
com.sun.star.frame.XComponentLoader xLoader =
 (com.sun.star.frame.XComponentLoader)UnoRuntime.queryInterface(
  com.sun.star.frame.XComponentLoader.class,
  xSMGR.createInstance("com.sun.star.frame.Desktop"));

// Load the document into the target frame by using our unambigous name
// and special search flags.
xDocument = xLoader.loadComponentFromURL(
 sURL, sTarget, com.sun.star.frame.FrameSearchFlag.CHILDREN, lProperties);

// dont forget to restore old frame name ...
xFrame.setName(sOldName);
```

The `loadComponentFromURL()` call returns a reference to a `com.sun.star.lang.XComponent` interface. The object belonging to this interface depends on the loaded component. If it is a component that only provides a component window, but not a controller, the returned component is this window. If it is an office component that provides a controller, the returned component is the controller or its model, if these is one. All Writer, Calc, Draw, Impress or Math documents in OpenOffice.org support a model, therefore the `loadComponentFromURL()` call returns it. The database and bibliography components however, return a controller, because they do not have a model.

Closing Documents

The `loadComponentFromURL()` returns a `com.sun.star.lang.XComponent` interface has previously been discussed. The return value is a reference to a `com.sun.star.lang.XComponent` interface, the corresponding object is a disposable component, and the caller must take care of lifetime problems. An `XComponent` supports the following methods:

```
void dispose ()
void addEventListener ( [in] com::sun::star::lang::XEventListener xListener)
void removeEventListener ( [in] com::sun::star::lang::XEventListener aListener)
```

In principle, there is a simple rule. The documentation of a `com.sun.star.lang.XComponent` specifies the objects that can own a component. Normally, a client using an `XComponent` is the owner of the `XComponent` and has the responsibility to dispose of it or it is not the owner. If it is not the owner, it may add itself as a `com.sun.star.lang.XEventListener` at the `XComponent` and not call `dispose()` on it. This type of `XEventListener` supports one method in which a component reacts upon the fact that another component is about to be disposed of:

```
void disposing ( [in] com::sun::star::lang::EventObject Source )
```

However, the frame, controller and model are interwoven tightly, and situations do occur in which there are several owners, for example, if there is more than one view for one model, or one of these components is in use and cannot be disposed of, for example, while a print job is running or a modal dialog is open. Therefore, developers must cope with these situations and remember a few things concerning the deletion of components.

Closing a document has two aspects. It is possible that someone else wants to close a document being currently worked on And you may want to close a component someone else is using at the same time. Both aspects are discussed in the following sections. A code example that closes a document is provided at the end of this section.

Reacting Upon Closing

The first aspect is that someone else wants to close a component for which you hold a reference. In the current version of OpenOffice.org, there are three possibilities.

- If the component is used briefly as a stack variable, you do not care about the component after loading, or you are sure there will be no interference, it is justifiable to load the component without taking further measures. If the user is going to close the component, let the reference go out of scope, or release the reference when no longer required.
- If a reference is used, but it is not necessary to react when it becomes invalid and the object supports `com.sun.star.uno.XWeak`, you can hold a weak reference instead of a hard reference. Weak references are automatically converted to `null` if the object they reference is going to be disposed. Because the generic frame implementation, and also the controllers and models of all standard document types implement XWeak, it is recommended to use it when possible.
- If a hard reference is held or you want to know that the component has been closed and the new situation has to be accommodated, add a `com.sun.star.lang.XEventListener` at the `com.sun.star.lang.XComponent` interface. In this case, release the reference on a `disposing()` notification.

Sometimes it is necessary to exercise more control over the closing process, therefore a new, optional interface `com.sun.star.util.XCloseable` has been introduced whichis supported in versions beyond 641. If the object you are referencing is a `com.sun.star.util.XCloseable`, register it as a `com.sun.star.util.XCloseListener` and throw a `com.sun.star.util.CloseVetoException` when prompted to close. Since `XCloseable` is specified as an optional interface for frames and models, do not assume that this interface is supported. It is possible that the code runs with a OpenOffice.org version where frames and models do not implement `XCloseable`. Therefore ,be prepared for the case when you receive null when you try to query `XCloseable`. The `XCloseable` interface is described in more detail below.

How to Trigger Closing

The second aspect – to close a view of a component or the entire viewable component *yourself* – is more complex. The necessary steps depend on how you want to treat modified documents. Besides you have to prepare for the new `com.sun.star.util.XCloseable` interface, which will be implemented in future versions of OpenOffice.org.



Although `XCloseable` is not supported in the current version of OpenOffice.org, you already have to check for this interface to write compatible code. Not checking for `XCloseable` will be illegal in future versions. If a component supports this interface, you must not use any closing procedure other than calling `close()` at that interface.

The following three diagrams show the decisions to be made when closing a frame or a document model. The important points are: if you expect modifications, you must either handle them using `com.sun.star.util.XModifiable` and `com.sun.star.frame.XStorable`, or let the user do the necessary interaction by calling `suspend()` on the controller. In any case, check if the frame or model is an `XCloseable` and prefer `com.sun.star.util.XCloseable:close()` over a call to `dispose()`. The first two diagrams illustrate the separate closing process for frames and models, the third diagram covers the actual termination of frames and models.

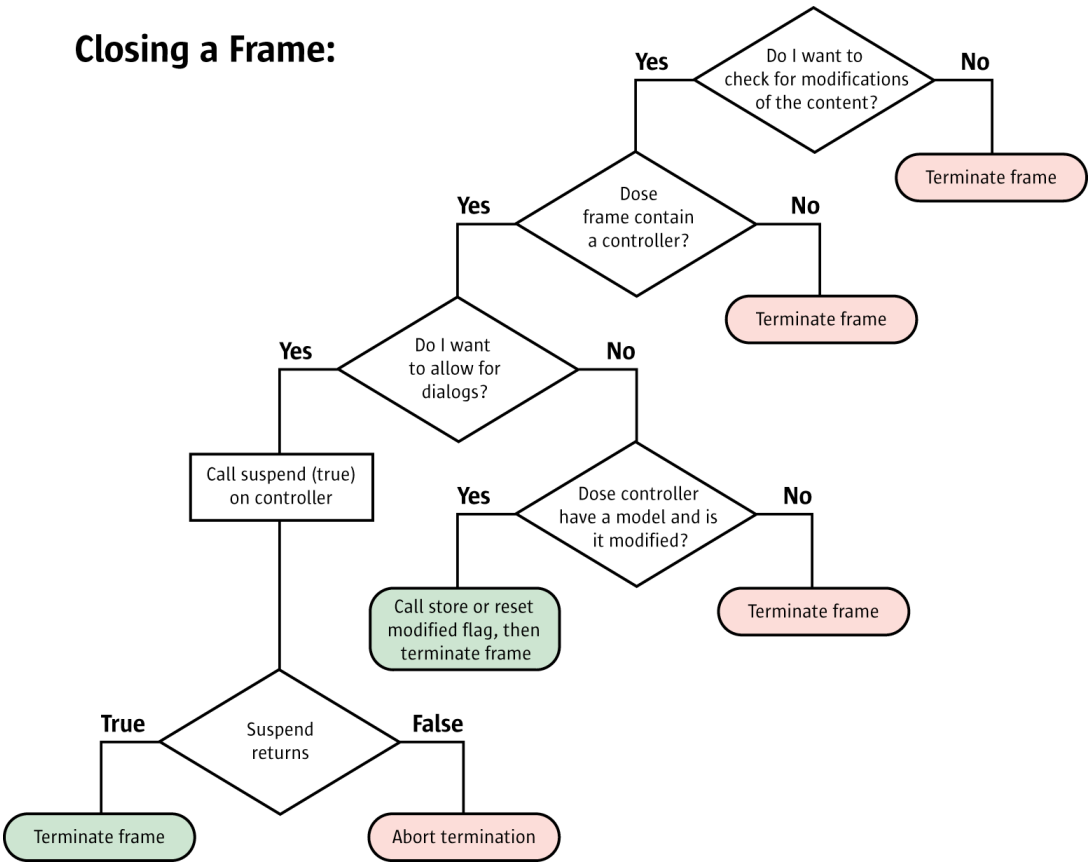


Illustration 60: Closing a Frame

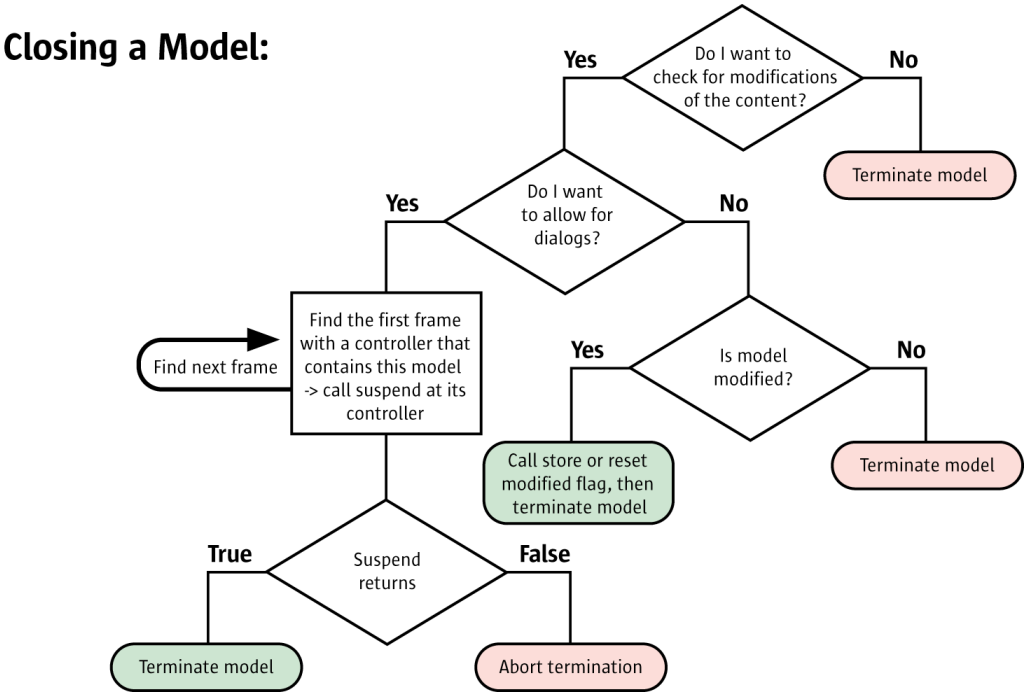


Illustration 61: Closing a Model

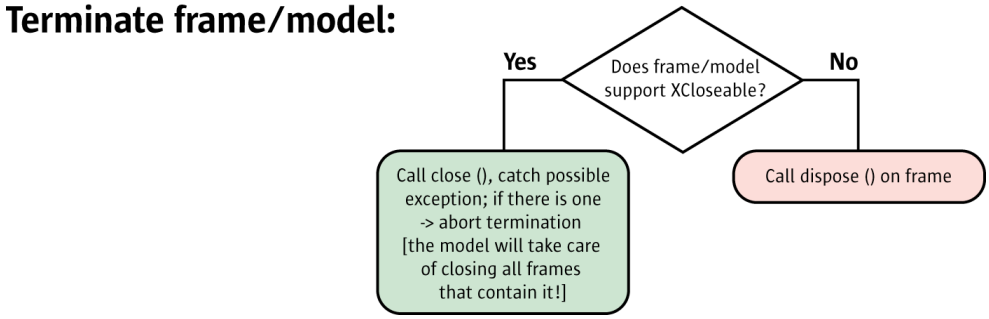


Illustration 62: Terminate Frame/Model

XCloseable

The dispose mechanism has shortcomings in complex situations, such as the frame-controller-model interaction. The dispose call cannot be rejected, but as shown above, sometimes it *is* necessary to prevent destruction of objects due to shared ownership or a state of the documents that forbids destruction.

A closing mechanism is required that enables all involved objects to negotiate if deletion is possible and to veto, if necessary. By offering the interface `com.sun.star.util.XCloseable`, a component tells it must be destroyed by calling `close()`. Calling `dispose()` on an `XCloseable` might lead to deadlocks or crash the entire application.

In OpenOffice.org, model or frame objects are possible candidates for implementing the interface `XCloseable`, therefore query for that interface before destroying the object. Call `dispose()` directly if the model or frame does not support the interface, thus declaring that it handles all the problems.

An object implementing `XCloseable` registers close listeners. When a close request is received, all listeners are asked for permission. If a listener wants to deprecate, it throws an exception derived from `com.sun.star.util.CloseVetoException` containing the reason why the component can not be closed. This exception is passed to the close requester. The `XCloseable` itself can veto the destruction by throwing an exception. If there is no veto, the `XCloseable` calls `dispose()` on itself and returns.

The `XCloseable` handles problems that occur if a component rejects destruction. A script programmer usually can not cope with a component not used anymore and refuses to be destroyed. Ensure that the component is destroyed to avoid a memory leak. The `close()` method offers a method to pass the responsibility to close the object to any possible close listener that vetoes closing or to the `XCloseable` if the initial caller is not able to stay in memory to try again later. This responsibility is referred to as *delivered ownership*. The mechanism sets some constraints on the possible reasons for an objection against a close request.

A close listener that is asked for permission can object for any reason if the close call does not force it to assume ownership of the closeable object.The close requester is aware of a possible failure. If the close call forces the ownership, the close listener must be careful. An objection is only allowed if the reason is temporary. As soon as the reason no longer exists, the owner automatically calls close on the object that should be closed, now being in the same situation as the initial close requester.

A permanent reason for objection is not allowed. For example,. the document is modified is not a valid reason to object, because it is unlikely that the document becomes unmodified by itself. Consequently, it could never be closed. Therefore, if an API programmer wants to avoid data loss, he must use the `com.sun.star.util.XModifiable` and `com.sun.star.frame.XStorable` inter-

faces of the document. The fact that a model refuses to be closed if it is modified is not dependable.

The interface `com.sun.star.util.XCloseable` inherits from `com.sun.star.util.XCloseBroadcaster` and has the following methods:

```
[oneway] void addCloseListener ( [in] com::sun::star::util::XCloseListener Listener );
[oneway] void removeCloseListener ( [in] com::sun::star::util::XCloseListener Listener );
void close ( [in] boolean DeliverOwnership )
```

The `com.sun.star.util.XCloseListener` is notified twice when `close()` is called on an `XClosable`:

```
void queryClosing ( [in] com::sun::star::lang::EventObject Source,
                    [in] boolean GetsOwnership )
void notifyClosing ( [in] com::sun::star::lang::EventObject Source )
```

Both `com.sun.star.util.XCloseable:close()` and `com.sun.star.util.XCloseListener:queryClosing()` throw a `com.sun.star.util.CloseVetoException`.

In the closing negotiations, an `XClosable` is asked to close itself. In the call to `close()`, the caller passes a boolean parameter `DeliverOwnership` to tell the `XClosable` that it will give up ownership in favor of an `XCloseListener`, or the `XCloseable` that might have to finish a job first, but will close the `XClosable` immediately when the job is completed.

After a call to `close()`, the `XClosable` notifies its listeners twice. First, it checks if itcan be closed. If not, it throws a CloseVetoException, otherwise it uses `queryClosing()` to see if a listener has any objections against closing. The value of `DeliverOwnership` is conveyed in the `GetsOwnership` parameter of `queryClosing()`. If no listener disapproves of closing, the `XClosable` exercises `notifyClosing()` on the listeners and disposes itself. The result of a call to `close()` on a *model* is that all frames, controllers and the model itself are destroyed. The result of a call to `close()` on a *frame* is that this frame is closed, but the model stays alive if there are other controllers.

If an `XCloseListener` does not agree on closing, it throws a `CloseVetoException`, and the `XClosable` lets the exception pass in `close()`, so that the caller receives the exception. The `CloseVetoException` tells the caller that closing failed. If the caller delegated its ownership in the call to `close()` by setting the `DeliverOwnership` parameter to true, an `XCloseListener` knows that it automatically assumes ownership by throwing a `CloseVetoException`. The caller knows that someone else is now the owner if it receives a `CloseVetoException`. The new owner is compelled to close the `XClosable` as soon as possible. If the `XCloseable` was the object that threw an exception, it is compelled also to close itself as soon as possible.



No API exists for trivial components. As a consequence, components are not allowed to do anything that prevents them from being destroyed. For example, since the office crashes when a container window or component window has an open modal dialog, every component that wants to open a modal dialog must implement the `com.sun.star.frame.XController` interface.

If a *model object* supports `XCloseable`, calling `dispose()` on it is forbidden, try to `close()` the `XCloseable` and catch a possible `CloseVetoException`. Components that cannot cope with a destroyed model add a close listener at the model. This enables them to object when the model receives a `close()` request. They also add as a close listener if they are not already added as an (dispose) event listener. This can be done by every controller object that uses that model. Tt is also possible to let the model iterate through its controllers and call their `suspend()` methods explicitly as a part of its implementation of the close method. It is only necessary to know that a method `close()` must be called to close the model with its controllers. The method the model chooses is an implementation detail.

The example below closes a loaded document component. It does not save modified documents or prompts the user to save. (OfficeDev/DesktopEnvironment/FunctionHelper.java)

```
// Conditions: xDocument = m_xLoadedDocument
 // Check supported functionality of the document (model or controller).
 com.sun.star.frame.XModel xModel =
  (com.sun.star.frame.XModel)UnoRuntime.queryInterface(
   com.sun.star.frame.XModel.class,xDocument);

 if(xModel!=null)
 {
  // It is a full featured office document.
  // Try to use close mechanism instead of a hard dispose().
  // But maybe such service is not available on this model.
  com.sun.star.util.XCloseable xCloseable =
    (com.sun.star.util.XCloseable)UnoRuntime.queryInterface(
     com.sun.star.util.XCloseable.class,xModel);


  if(xCloseable!=null)
  {
    try
    {
      // use close(boolean DeliverOwnership)
      // The boolean parameter DeliverOwnership tells objects vetoing the close process that they may
      // assume ownership if they object the closure by throwing a CloseVetoException
      // Here we give up ownership. To be on the safe side, catch possible veto exception anyway.
      xCloseable.close(true);
    }
    catch(com.sun.star.util.CloseVetoException exCloseVeto)
    {
    }
  }
  // If close is not supported by this model - try to dispose it.
  // But if the model disagree with a reset request for the modify state
  // we shouldn't do so. Otherwhise some strange things can happen.
  else
  {
      com.sun.star.lang.XComponent xDisposeable =
        (com.sun.star.lang.XComponent)UnoRuntime.queryInterface(
         com.sun.star.lang.XComponent.class,xModel);
         xDisposeable.dispose();
       }
       catch(com.sun.star.beans.PropertyVetoException exModifyVeto)
       {
       }
     }
   }
 }
```

Storing Documents

After loading an office component successfully, the returned interface cis used to manipulate the component. Document specific interfaces, such as the interfaces `com.sun.star.text.XTextDocument`, `com.sun.star.sheet.XSpreadsheetDocument` or `com.sun.star.drawing.XDrawPagesSupplier` are retrieved using `queryInterface()`.

If the office component supports the `com.sun.star.frame.XStorable` interface applying to every component implementing the service `com.sun.star.document.OfficeDocument`, it can be stored:

```
void store ( )
void storeAsURL ( [in] string sURL,
                  [in] sequence< com::sun::star::beans::PropertyValue > lArguments )
void storeToURL ( [in] string sURL,
                  [in] sequence< com::sun::star::beans::PropertyValue > lArguments )
boolean hasLocation ()
string getLocation ()
boolean isReadonly ()
```

The `XStorable` offers the methods `store()`, `storeAsURL()` and `storeToURL()` for storing. The latter two methods are called with a media descriptor.

The method `store()` overwrites an existing file. Calling this method on a document that was created from scratch using a *private:factory/...* URL leads to an exception.

The other two methods `storeAsURL()` and `storeToURL()` leave the original file untouched and differ after the storing procedure. The `storeToURL()` method saves the current document to the desired location without touching the internal state of the document. The method `storeAsURL` sets

the `Modified` attribute of the document, accessible through its `com.sun.star.util.XModifiable` interface, to `false` and updates the internal media descriptor of the document with the parameters passed in the call. This changes the document URL.

The following example exports a Writer document, Writer/Web document or Calc sheet to HTML. (OfficeDev/DesktopEnvironment/FunctionHelper.java)

```
// Conditions: sURL      = "file:///home/target.htm"
//             xDocument = m_xLoadedDocument

// Export can be achieved by saving the document and using
// a special filter which can write the desired format.
// Normally this filter should be searched inside the filter
// configuration (using service com.sun.star.document.FilterFactory)
// but here we use well known filter names directly.

String sFilter = null;

// Detect document type by asking XServiceInfo
com.sun.star.lang.XServiceInfo xInfo = (com.sun.star.lang.XServiceInfo)UnoRuntime.queryInterface (
  com.sun.star.lang.XServiceInfo .class, xDocument);

// Determine suitable HTML filter name for export.
if(xInfo!=null)
{
 if(xInfo.supportsService ("com.sun.star.text.TextDocument ") == true)
     sFilter = new String("HTML (StarWriter) ");
 else
 if(xInfo.supportsService ("com.sun.star.text.WebDocument ") == true)
     sFilter = new String("HTML ");
 else
 if(xInfo.supportsService ("com.sun.star.sheet.SpreadsheetDocument ") == true)
     sFilter = new String("HTML (StarCalc) ");
}

if(sFilter!=null)
{
 // Build necessary argument list for store properties.
 // Use flag "Overwrite" to prevent exceptions, if file already exists.

 com.sun.star.beans.PropertyValue[] lProperties =
     new com.sun.star.beans.PropertyValue[2];
 lProperties[0]       = new com.sun.star.beans.PropertyValue();
 lProperties[0].Name  = "FilterName ";
 lProperties[0].Value = sFilter;
 lProperties[1]       = new com.sun.star.beans.PropertyValue();
 lProperties[1].Name  = "Overwrite ";
 lProperties[1].Value = new Boolean(true);

 com.sun.star.frame.XStorable xStore = (com.sun.star.frame.XStorable)UnoRuntime.queryInterface (
     com.sun.star.frame.XStorable .class, xDocument);

 xStore.storeAsURL (sURL, lProperties);
}
```

If a model is loaded or stored successfully, all parts of the media descriptor not explicitly excluded according to the media descriptor table in section *6.1.5 Office Development - OpenOffice.org Application Environment - Handling Documents - Loading Documents - MediaDescriptor* must be provided by the methods `getURL()` and `getArgs()` in the `com.sun.star.frame.XModel` interface. The separation of the URL and the other arguments is used, because the URL is the often the most wanted part for itsperformance optimized access.



The XModel offers a method `attachResource()` that changes the media descriptor of the document, but this method should only be used in special cases, for example, by the implementer of a new document model and controller. The method `attachResource()` does not force reloading of the document. Validation checks are done when a document is loaded through MediaDescriptor. For example, if the resource is write protected, add `Readonly` to the MediaDescriptor and the filter name must match the data. A possible use for `attachResource()` could be creating a document from a template, where after loading successfully, the document's resource is changed to an "unnamed" state by deleting the URL.

Printing Documents

Printing revolves around the interface `com.sun.star.view.XPrintable`. Its methods and special printing features for the various document types are described in the document chapters *7.2.3 Text Documents - Handling Text Document Files - Printing Text Documents*, *8.2.3 Spreadsheet Documents - Handling Spreadsheet Document Files - Printing Spreadsheet Documents*, *9.2.3 Drawing - Handling Drawing Document Files - Printing Drawing Documents* and *9.4.2 Drawing - Handling Presentation Document Files - Printing Presentation Documents*.

6.1.6 Using the Dispatch Framework

The component framework with the Frame-Controller-Model paradigm builds the skeleton of the global object structure. Other frameworks are defined that enrich the communication between an office component and the desktop environment. Usually they start at a frame object for the frame anchors an office component in the desktop environment.

One framework is the dispatch framework. Its main purpose defines interfaces for a *generic communication* between an office component and a user interface. This *communication* process handles requests for command executions and gives information about the various attributes of an office component. *Generic* means that the user interface does not have to know all the interfaces supported by the office component.The user interfaces sends messages to the office component and receives notifications.Tthe messages use a simple format. The entire negotiation about supported commands and parameters can happen at runtime while an application built on the specialized interfaces of the component are created at compile or interpret time. This generic approach is achieved by looking at an office component differently, not as objects with method-based interfaces, but as slot machines that take standardized command tokens.

We have discussed the differences between the different document types. The common functionality covers the generic features, that is, an office component is considered to be the entirety of its controller, its model and *many* document-specific interfaces. To implement a user interface for a component, it would be closely bound to the component and its specialized interfaces. If different components use different interfaces and methods for their implementations, similar functions cannot be visualized by the same user interface implementation. For instance, an action like **Edit – Select All** leads to different interface calls depending on the document type it is sent to. From a user interface perspective, it would be better to define abstract descriptions of the actions to be performed and let the components decide how to handle these actions, or not to handle . These abstract descriptions and how to handle them is specified in the dispatch framework.

Command URL

In the dispatch framework, every possible user action is defined as an executable *command*, and every possible visualization as a reflection of something that is exposed by the component is defined as an *attribute*. Every executable command and every attribute is a feature of the office component, and the dispatch framework gives every feature a name called *command URL*. It is represented by a `com.sun.star.util.URL` struct.

Command URLs are strings that follow the *protocol_scheme:protocol_specific_part* pattern. Public URL schemes, such as *file:* or *http* can be used here.Executing a request with a URL that points to a location of a document means that this document is loaded. In general, both parts of the command URL can be arbitrary strings, but a request cannot be executed if there is an object that does not know how to handle its command URL.

Processing Chain

A request is created by any object.User interface objects can create requests. Consider a toolbox where different functions acting on the office component are presented as buttons. When a button is clicked, the desired functionality is executed. If the code assigned to the button is provided with a suitable command URL, it handles the user action by creating the request and finding a component that can handle it. The button handler does not require any prior knowledge of the component and how it would go about its task.

This situation is handled by the design pattern *chain of responsibility*. Everything a component needs to know to execute a request is the last link of a chain of objects capable of executing requests. If this object gets the request, it checks if it can handle it or passes it to the next chain member until the request is executed, or the end of the chain is reached.

The chain members in the dispatch framework are objects implementing the interface `com.sun.star.frame.XDispatchProvider`. Every frame and controller supports it.In the simplest case, the chain consists of two members, a frame and its controller, but concatenating several chain parts on demand of a frame or a controller is possible. A controller once called, passes on the call, that is, it can use internal frames created by its implementation. A frame also passes the call to other objects, for example, its parent frame.

The current implementation of the chain is different from a simple chain.A frame is always the leading chain member and *must* be called initially, but in the default implementation used in OpenOffice.org, the frame first(!) asks its controller before it goes on with the request. Other frame implementations handle this in a different way. Other chain members are inserted into the call sequence before the controller uses the dispatch interception capability of a frame. The developers should not rely on any particular order inside the chain.

The dispatch framework uses a generic approach to describe and handle requests with a lose coupling between the participating objects. To work correctly, it is necessary to follow certain rules:

1. Every chain starts at a frame, and this object decides if it passes on the call to its controller. The controller is not called directly from the outside. This is not compulsory for internal usage of the dispatch API inside an office component implementation. Ther two reasons for this rule are:
   - A frame providesa `com.sun.star.frame.XDispatchProviderInterception` interface, where other dispatch providers dock. The frame implementation guarantees that these interceptors are called before the frame handles the request or passes it to the controller. This allows a sophisticated customization of the dispatch handling.
   - If a component is placed into a context where parts of its functionality are not be exposed to the outside, a special frame implementation is used to suppress or handle requests before they are passed to the controller. This frame can add or remove arguments to requests and exchange them.
2. A command URL isparsed into a `com.sun.star.util.URL` struct before passing it to a dispatch provider, because it is assumed that the call is passed on to several objects. Having a preparsed URL saves parsing the command string repeatedly. Parsing means that the members `Complete`, `Main`, `Protocol` and at least one more member of the `com.sun.star.util.URL` struct, depending on the given protocol scheme have to be set. Additional members are set if the concrete URL protocol supports them. For well known protocol schemes and protocol schemes specific to OpenOffice.org, the service `com.sun.star.util.URLTransformer` is used to fill the struct from a command URL string. For other protocols, the members are set explicitly, but it is also possible to write an extended version of the `URLTransformer` service to carry out URL parsing. An extended `URLTransformer` must support all protocols supported by the

default URLTransformer implementation, for example, by instantiating the old implementation by its implementation name and forwarding all known URLs to it, except URLs with new protocols.

The dispatch framework connects an object that creates a request with another object that reacts on the request. In addition, it provides feedback to the requester. It can tell if the request is currently allowed or not. If the request acts on a specific attribute of an object, it c provides the current status of this attribute. Altogether, this is called *status information*, represented by a `com.sun.star.frame.FeatureStateEvent` struct. This information is reflected in a user interface by enabling or disabling controls to show their availability, or by displaying the status of objects. For example, a pressed button for the bold attribute of text, or a numeric value for the text height in a combo box.

The `com.sun.star.frame.XDispatchProvider` interface does not handle requests, but delegates every request to an individual dispatch object implementing `com.sun.star.frame.XDispatch`.



This is the concept, but the implementation is not forced and it may decide to return the same object for every request. It is not recommened to use the dispatch provider object as a dispatch object.

Dispatch Process

This section describes the necessary steps to handle dispatch providers and dispatch objects. The illustration below shows the services and interfaces of the the Dispatch framework.

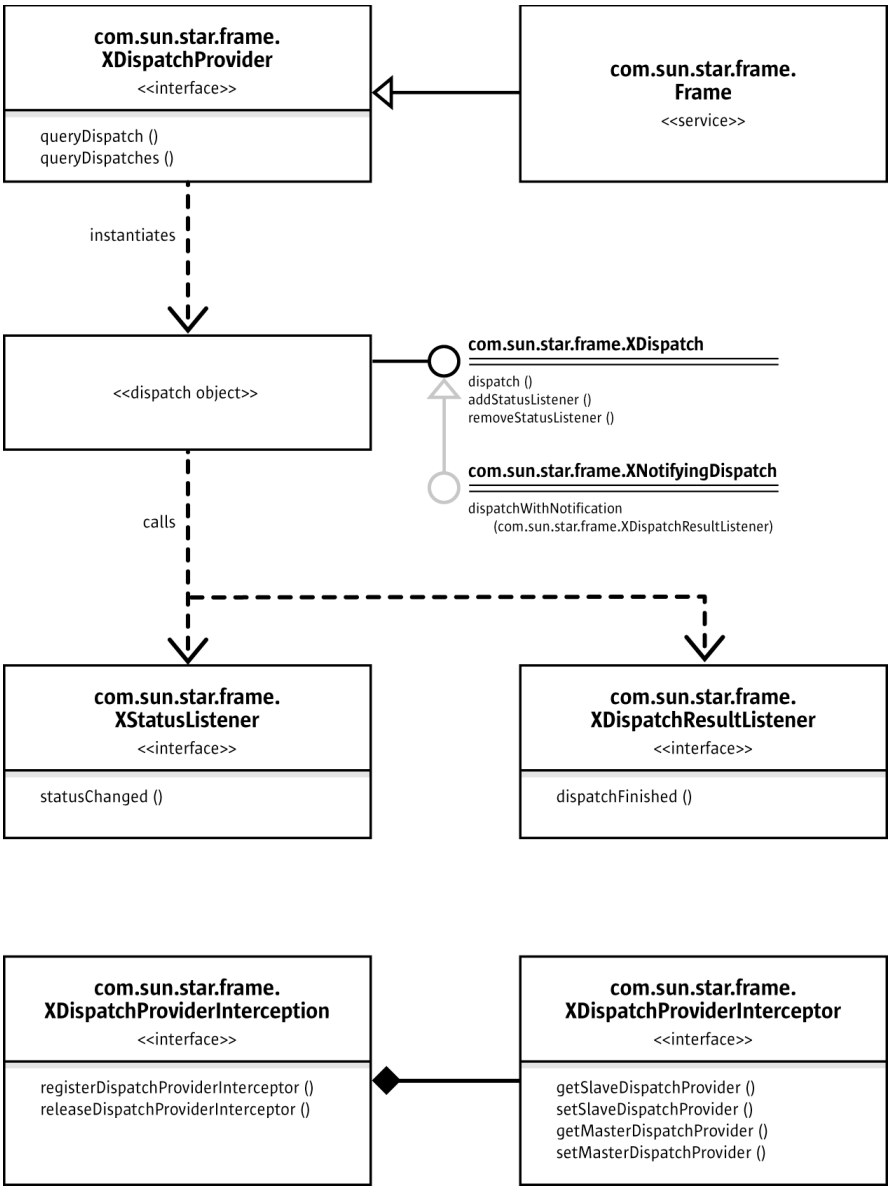


Illustration 63: Dispatch Framework

Getting a Dispatch Object

First, create a command URL that represents the desired functionality ensuring that it is parsed as described above. Tables with possible command URLs for the default office components of OpenOffice.org are located in the appendix.

Request the `com.sun.star.frame.XDispatchProvider` interface of the frame that contains the office component for a dispatch object for the command URL by calling its `queryDispatch()` method.

```
com::sun::star::frame::XDispatch queryDispatch ( [in] com::sun::star::util::URL URL,
                                 [in] string TargetFrameName,
                                 [in] long SearchFlags )
sequence< com::sun::star::frame::XDispatch > queryDispatches (
                               [in] sequence< com::sun::star::frame::DispatchDescriptor > Requests )
```

The additional parameters (`TargetFrameName`, `SearchFlags`) of this call are only used for dispatching public URL schemes, because they specify a target frame and frame search mode to the loading process. Valid target names and search flags are described in the section *6.1.5 Office Development - OpenOffice.org Application Environment - Handling Documents - Loading Documents - Target Frame*. The targets `"_self"`, `"_parent"` and `"_top"` are well defined, so that they can be used, because a `queryDispatch()` call starts at a frame object. Using frame names or search flags with command URLs does not have any meaning in the office components in OpenOffice.org.

You receive a dispatch object that supports at least `com.sun.star.frame.XDispatch`:

```
[oneway] void dispatch ( [in] com::sun::star::util::URL URL,
                         [in] sequence< com::sun::star::beans::PropertyValue > Arguments )
[oneway] void addStatusListener ( [in] com::sun::star::frame::XStatusListener Control,
                          [in] com::sun::star::util::URL URL )
[oneway] void removeStatusListener ( [in] com::sun::star::frame::XStatusListener Control,
                          [in] com::sun::star::util::URL URL )
```

Listening for Status Information

If a dispatch object is received, add a listener for status events by calling its `addStatusListener()` method. A `com.sun.star.frame.XStatusListener` implements:

```
[oneway] void statusChanged ( [in] com::sun::star::frame::FeatureStateEvent Event )
```

Keep a reference to the dispatch object until you call the `removeStatusListener()` method, because it is not sure that any other object will keep it alive. If a status listener is not registered, because you want to dispatch a command,and are not interested in status events, release all references to the dispatch object immediately after usage. If a dispatch object is not received, the desired functionality is not available. If you have a visual user interface element that represents that functionality, disable it.

If a status listener is registered and there is status information, a `com.sun.star.frame.FeatureStateEvent` is received immediately after registering the listener. Status information is still received later if the status changes and you are still listening. The `IsEnabled` member of the `com.sun.star.frame.FeatureStateEvent` tells you if the functionality is currently available, and the `State` member holds information about a status that could be represented by UI elements. Its type depends on the command URL. A boolean status information is visualized in a pressed or not pressed look of a toolbox button. Other types need complex elements, such as combo boxes or spinfields embedded in a toolbox that show the current font and font size. If the `State` member is empty, the action does not have an explicit status, such as the menu item **File – Print.** The current status can be ambiguous, because more than one object is selected and the objects are in a different status, for example. selected text that is partly formatted bold and partly regular.

A special event is a status event where the `Requery` flag is set. This is a request to release all references to the dispatch object and to ask the dispatch provider for a new object, because the old one has become invalid. This allows the office components to accommodate internal context changes. It is possible that a dispatch object is not received, because the desired functionality has become unavailable.

If you do not get any status information in your `statusChanged()` implementation, assume that the functionality is always available, but has no explicit status.

If you are no longer interested in status events, use the `removeStatusListener()` method and release all references to the dispatch object. You may get a `disposing()` callback from the dispatch object when it is going to be destroyed. It is not necessary to call `removeStatusListener()`. Ensure that you do not hold any references to the dispatch object anymore.

Listening for Context Changes

Sometimes internal changes, for example, travelling from a text paragraph to a text table, or selecting a different type of object, force an office component to invalidate all referenced dispatch objects and provides other dispatch objects, including dispatches for command URLs it could not handle before. The component then calls the `contextChanged()` method of its frame, and the frame broadcasts the corresponding `com.sun.star.frame.FrameActionEvent`. For this reason, register a frame action listener using `addFrameActionListener()` at frames you want dispatch objects. Refer to section *6.1.3 Office Development - OpenOffice.org Application Environment - Using the Component Framework - Frames - XFrame - Frame Actions* for additional information. If the listener is called back with a CONTEXT_CHANGED event, release all dispatch objects and query new dispatch objects for every command URL you require. You can also try command URLs that did not get a dispatch object before.

If you are no longer interested in context changes of a frame, use the `removeFrameActionListener()` method of the frame to deregister and release all references to the frame. If you get a `disposing()` request from the frame in between, it is not necessary to call `removeFrameActionListener()`, but you must release all frame references you are currently holding.

Dispatching a Command

If the desired functionality is available, execute it by calling the `dispatch()` method of the dispatch object. This method is called with the same command URL you used to get it, and optionally with a sequence of arguments of type `com.sun.star.beans.PropertyValue` that depend on the command. It is not redundant that supplied the URL again, because it is allowed to use one dispatch object for many command URLs. The appendix shows the names and types for the parameters. However, the command URLs for simple user interface elements, such as menu entries or toolbox buttons send no parameters. Complex user interface elements use parameters, for example, a combo box in a toolbar that changes the font height. (OfficeDev/DesktopEnvironment/FunctionHelper.java)

```
 // Conditions: sURL        = "private:factory/swriter"
 //             lProperties = new com.sun.star.beans.PropertyValue[0]
 //             xSMGR       = m_xServiceManager
 //             xListener   = this
//             xFrame      = a given frame

 // Query the frame for right interface which provides access to all
 // available dispatch objects.
 com.sun.star.frame.XDispatchProvider xProvider =
    (com.sun.star.frame.XDispatchProvider)UnoRuntime.queryInterface (
        com.sun.star.frame.XDispatchProvider .class, xFrame );

// Create and parse a valid URL
 // Note: because it is an in/out parameter we must use an array of URLs
 com.sun.star.util.XURLTransformer xParser =
    (com.sun.star.util.XURLTransformer)UnoRuntime.queryInterface (
        com.sun.star.util.XURLTransformer .class,
            xSMGR.createInstance("com.sun.star.util.URLTransformer" ));

 com.sun.star.util.URL[] aParseURL = new com.sun.star.util.URL[1];
 aParseURL[0]          = new com.sun.star.util.URL();
 aParseURL[0].Complete = sURL;

xParser.parseStrict (aParseURL);

 // Ask for dispatch object for requested URL and use it.
 // Force given frame as target "" which means the same like "_self".
 xDispatcher = xProvider.queryDispatch(aParseURL[0],"",0);

if(xDispatcher!=null)
 {
  xDispatcher.addStatusListener (xListener,aParseURL[0]);
  xDispatcher.dispatch (aParseURL[0],lProperties);
 }
```

Dispatch Results

Every dispatch object implement optional interfaces. An important extension is the `com.sun.star.frame.XNotifyingDispatch` interface for dispatch results. The `dispatch()` call is a `void` method and should be treated as an asynchronous or oneway call, therefore a dispatch result can not be passed as a return value, rather, a callback interface is necessary. The interface that provides dispatch results by a callback is the `com.sun.star.frame.XNotifyingDispatch` interface:

```
[oneway] void dispatchWithNotification ( [in] com::sun::star::util::URL URL,
                          [in] sequence< com::sun::star::beans::PropertyValue > Arguments,
                          [in] com::sun::star::frame::XDispatchResultListener Listener )
```

Its method `dispatchWithNotification()` takes a `com.sun.star.frame.XDispatchResultListener` interface that is called after a dispatched URL has been executed.



Although the dispatch process is considered to be asynchronous, this is not necessarily so. Therefore, be prepared to get the dispatch result notification before the dispatch call returns.

The dispatch result is transferred as a `com.sun.star.frame.DispatchResultEvent` struct in the callback method `dispatchFinished()`. The `State` member of this struct tells if the dispatch was successful or not, while the `Result` member contains the value that would be returned if the call had been executed as a synchronous function call. The appendix shows the types of return values. If a public URL is dispatched, the dispatch result is a reference to the frame the component was loaded into. (OfficeDev/DesktopEnvironment/FunctionHelper.java)

```
// Conditions: sURL        = "private:factory/swriter"
//             lProperties = new com.sun.star.beans.PropertyValue[0]
//             xSMGR       = m_xServiceManager
//             xListener   = this

// Query the frame for right interface which provides access to all
// available dispatch objects.
com.sun.star.frame.XDispatchProvider xProvider =
   (com.sun.star.frame.XDispatchProvider)UnoRuntime.queryInterface (
       com.sun.star.frame.XDispatchProvider .class, xFrame);

// Create and parse a valid URL
// Note: because it is an in/out parameter we must use an array of URLs
com.sun.star.util.XURLTransformer xParser =
   (com.sun.star.util.XURLTransformer)UnoRuntime.queryInterface(
      com.sun.star.util.XURLTransformer .class,
          xSMGR.createInstance ("com.sun.star.util.URLTransformer "));

// Ask for right dispatch object for requested URL and use it.
// Force given frame as target "" which means the same like "_self".
// Attention: The interface XNotifyingDispatch is an optional one!
com.sun.star.frame.XDispatch   xDispatcher =
   xProvider.queryDispatch (aURL,"",0);

com.sun.star.frame.XNotifyingDispatch xNotifyingDispatcher =
   (com.sun.star.frame.XNotifyingDispatch)UnoRuntime.queryInterface (
       com.sun.star.frame.XNotifyingDispatch.class , xDispatcher );

if(xNotifyingDispatcher!=null)
   xNotifyingDispatcher.dispatchWithNotification (aURL, lProperties, xListener);
```

Dispatch Interception

The dispatch framework described in the last chapter establishes a communication between a user interfaces and an office component. Both can be OpenOffice.org default components or custom components. Sometimes it is not necessary to replace a UI element by a new implementation. It can be sufficient to influence its visualized state or to redirect user interactions to external code. This is the typical use for dispatch interception.

The dispatch communication works in two directions: status information is transferred from the office component to the UI elements and user requests travel from the UI element to the office component. Both go through the same switching center that is, an object implementing `com.sun.star.frame.XDispatch`. The UI element gets this object by calling `queryDispatch()` at the frame containing the office component, and usually receives an object that connects to code inside the frame, the office component or global services in OpenOffice.org. The frame offers an interface that is used to return third-party dispatch objects that provide the UI element with status updates. For example, it is possible to disable a UI element that would not be disabled otherwise. Another possibility is to write replacement code that is called by the UI element if the user performs a suitable action.

Dispatch objects are provided by objects implementing the `com.sun.star.frame.XDispatchProvider` interface, and that is the interface you are required to implement. There is an extra step where the dispatch provider must be attached to the frame to intercept the dispatching communication, therefore the dispatch provider becomes a part of the *chain of responsibility* described in the previous section. This is accomplished by implementing `com.sun.star.frame.XDispatchProviderInterceptor`.

This chain usually only consists of the frame and the controller of the office component it contains, but the frame offers the `com.sun.star.frame.XDispatchProviderInterception` interface where other providers are inserted. They are called before the frame tries to find a dispatch object for a command URL, so that it is possible to put the complete dispatch communication in a frame under external control. More than one interceptor can be registered, thus building a bigger chain.

Routing every dispatch through the whole chain becomes a performance problem, because could be more than a hundred possible clients asking for a dispatch object. For this reason there is also an API that limits the routing procedure to particular commands or command groups. This is described below.

Once the connection is established, the dispatch interceptor decides how requests for a dispatch object are dealt with. When asked for a dispatch object for a Command URL, it can:

- Return an empty interface that disables the corresponding functionality.

  There's a bug in Ooo1.0/SO6.0 that this does not work, so disabling must be done explicitly (see below). It will be fixed in Ooo1.02/SO6.02.

- Pass the request to the next chain member, called *slave dispatcher provider* described below if it is not interested in that functionality.

- Handle the request and return an object implementing `com.sun.star.frame.XDispatch`. As described in the previous chapter, client objects may register at this object as status event listeners. The dispatch object returns any possible status information as long as the type of the "State" member in the `com.sun.star.frame.FeatureStateEvent` struct has one of the expected types, otherwise the client requesting the status information can not handle it properly. The expected types must be documented together with the existing commands.For example, if a menu entry wants status information, it handles a void, that is, do nothing special or a boolean state by displaying a check mark, but nothing else.
  The status information could contain a `disable` directive. Note that a dispatch object returns status information immediately when a listener registers. Any , events change can be broadcasted at arbitrary points in time.

- The returned dispatch object is also used by client objects to dispatch the command that matches the command URL. The dispatch object receiving this request checks if the code it wants to execute is valid under the current conditions. It is not sufficient to rely on `disable` requests, because a client is not forced to register as a status listener if it wants to dispatch a request.

The *slave dispatch provider* and *master dispatch provider* in the `com.sun.star.frame.XDispatchProviderInterceptor` interface are a bit obscure at first. They are two pointers to chain members in both directions, next and previous, where the first and last member in the chain have special meanings and responsibilities.

The command dispatching passes through a chain of dispatch providers, starting at the frame. If the frame is answered to include an interceptor in this chain, the frame inserts the interceptor in the chain and passes the following chain member to the new chain member, so that calls are passed along the chain if it does not want to handle them.

If any interceptor is deregistered, the frame puts the lose ends together by adjusting the master and slave pointer of the chain successor and predecessor of the element that is going to be removed from the chain. All of them are interceptors, so only the last slave is a dispatch provider.

The frame takes care of the whole chain in the register or deregister of calls in the dispatch provider interceptor, so that the implementer of an interceptor does not have to be concerned with the chain construction.

6.1.7 Java Window Integration

This section discusses experiences obtained during the development of Java-OpenOffice.org integration. Usually, developers use the OfficeBean for this purpose. The following provides background information about possible strategies to reach this goal.

There are multiple possibilities to integrate local windows with OpenOffice.org windows. This chapter shows the integration of OpenOffice.org windows into a Java bean environment. Some of this information maybe helpful with other local window integrations.

The Window Handle

An important precondition is the existence of a system window handle of the own Java window. For this, use a `java.awt.Canvas` and the following JNI methods:

- a method to query the window handle (HWND on Windows, X11 ID on UNIX)
- a method to identify the operating system, for example, UNIX, Windows, or Macintosh

For an example, see *bean/com/sun/star/beans/LocalOfficeWindow.java*

The two methods `getNativeWindow()` and `getNativeWindowSystemType()` are declared and exported, but implemented for windows in *bean/native/win32/com_sun_star_beans_LocalOfficeWindow.c* through JNI



It has to be a `java.awt.Cavans`. These JNI methods cannot be implemented at a Swing control, because it does not have its own system window. You can use a `java.awt.Canvas` in a Swing container environment.



The handle is not available before the window is visible, otherwise the JNI function does not work. One possibility is to cache the handle and set it in `show()` or `setVisible()`.

Using the Window Handle

The window handle create the OpenOffice.org window. There are two ways to accomplish this:

A Hack

This option is mentioned because there are situations where this is the only feasible method. The knowledge of this option can help in other situations.

Add the UNO interface `com.sun.star.awt.XWindowPeer` so that it is usable for the OpenOffice.org window toolkit. This interface can have an empty implementation. In `com.sun.star.awt.XToolkit:createWindow()`, another interface `com.sun.star.awt.XSystemDependentWindowPeer` is expected that queries the HWND. Thus, `XWindowPeer` is for transporting and `com.sun.star.awt.XSystemDependentWindowPeer` queries the HWND.

This method getsa `com.sun.star.awt.XWindow` as a child of your own Java window, that is used to initialize a `com.sun.star.frame.XFrame`.
(OfficeDev/DesktopEnvironment/FunctionHelper.java)

```
com.sun.star.awt.XToolkit xToolkit =
          (com.sun.star.awt.XToolkit)UnoRuntime.queryInterface(
          com.sun.star.awt.XToolkit.class,
          xSMGR.createInstance("com.sun.star.awt.Toolkit"));

// this is the canvas object with the JNI methods
 aParentView = ...
 // some JNI methods cannot work before this
aParentView.setVisible(true);

// now wrap the canvas (JavaWindowPeerFake) and add the necessary interfaces
com.sun.star.awt.XWindowPeer xParentPeer =
          (com.sun.star.awt.XWindowPeer)UnoRuntime.queryInterface(
              com.sun.star.awt.XWindowPeer.class,
              new JavaWindowPeerFake(aParentView));

com.sun.star.awt.WindowDescriptor aDescriptor = new com.sun.star.awt.WindowDescriptor();
aDescriptor.Type                =   com.sun.star.awt.WindowClass.TOP;
aDescriptor.WindowServiceName   =   "workwindow";
aDescriptor.ParentIndex         =   1;
aDescriptor.Parent              =   xParentPeer;
aDescriptor.Bounds              =   new com.sun.star.awt.Rectangle(0,0,0,0);
if (aParentView.getNativeWindowSystemType()==com.sun.star.lang.SystemDependent.SYSTEM_WIN32)
          aDescriptor.WindowAttributes = com.sun.star.awt.WindowAttribute.SHOW;
else
          aDescriptor.WindowAttributes = com.sun.star.awt.WindowAttribute.SYSTEMDEPENDENT;

// now the toolkit can create an com.sun.star.awt.XWindow
 com.sun.star.awt.XWindowPeer xPeer = xToolkit.createWindow( aDescriptor );
com.sun.star.awt.XWindow xWindow =
          (com.sun.star.awt.XWindow)UnoRuntime.queryInterface(
          com.sun.star.awt.XWindow.class,
          xPeer);
```

Legal Solution

The `com.sun.star.awt.Toolkit` service has a method `com.sun.star.awt.XSystemChildFactory` with a method `createSystemChild()`. This accepts an `any` with a wrapped HWND or X Window ID, as long and the system type, such as Windows, Java, and UNIX directly. Here you create an `com.sun.star.awt.XWindow`. This method cannot be used in OpenOffice.org build versions before src642, because the process ID parameter is unknown to the Java environment. Newer versions do not check this parameter, thus this new, method works.



As a user of `com.sun.star.awt.XSystemChildFactory:createSystemChild()` ensure that your client (Java application) and your server (OpenOffice.org) use the same display. Otherwise the window handle is not interchangeable.

(OfficeDev/DesktopEnvironment/FunctionHelper.java)

```
com.sun.star.awt.XToolkit xToolkit =
          (com.sun.star.awt.XToolkit)UnoRuntime.queryInterface(
              com.sun.star.awt.XToolkit.class,
              xSMGR.createInstance("com.sun.star.awt.Toolkit"));
```

```
// this is the canvas with the JNI functions
 aParentView = ...
 // some JNI funtions will not work withouth this
aParentView.setVisible(true);

// no wrapping necessary, simply use the HWND
com.sun.star.awt.XSystemChildFactory xFac =
         (com.sun.star.awt.XSystemChildFactory)UnoRuntime.queryInterface(
            com.sun.star.awt.XSystemChildFactory.class,
            xToolkit);

Integer nHandle = aParentView.getHWND();
byte[] lIgnoredProcessID = new byte[0];

com.sun.star.awt.XWindowPeer xPeer =
         xFac.createSystemChild(
                  (Object)nHandle,
                  lIgnoredProcessID,
                  com.sun.star.lang.SystemDependent.SYSTEM_WIN32);

com.sun.star.awt.XWindow xWindow =
         (com.sun.star.awt.XWindow)UnoRuntime.queryInterface(
            com.sun.star.awt.XWindow.class,
            xPeer);
```



The old method still works and can be used, but it should be considered deprecated. If in doubt, implement both and try the new method at runtime. If it does not work, try the hack.

Resizing

Another difficulty is resizing the window. Normally, the child window expects resize events of the parent. The child does not resize it window, because it must know the layout of the parent window. The VCL, OpenOffice.org's windowing engine creates a special system child window, thus we can resize windows.

The parent window can be filled "full size" with the child window, but only for UNIX and not for Windows. The VCL's implementation is system dependent.

The bean deals with this issue by adding another function to the local library. Windows adds arbitrary properties to an HWND. You can also subclass the window, that is, each Windows window has a function pointer or callback to the function that performs the event handling (WindowProcedure). Using this, it is possible to treat events by calling your own methods. This is useful whenever the window is not created by you and you need to influence the behavior of the window.

In this case, the Java window has not been created by us, but we need to learn about resize events to forward these to the OpenOffice.org window. Look at the file *bean/native/win32/com_sun_star_beans_LocalOfficeWindow.c*, and find the method `OpenOfficeWndProc()`. In the first call of the JNI function `Java_com_sun_star_beans_LocalOfficeWindow_getNativeWindow()` of this file, the own handler is applied to the foreign window.



The old bean implementation had a bug that is fixed in newer versions. If you did not check if the function pointer was set, and called `Java_com_sun_star_beans_LocalOfficeWindow_getNativeWindow()` multiple times, you created a chain of functions that called each other with the result of an endless recursion leading to a stack overflow. If the own handler is already registered, it is now marked in one of the previously mentioned properties registered with an HWND:

In the future, VCL will do this sub-classing by itself, even on Windows. This will lead to equal behavior between Windows and UNIX.

The initial size of the window is a related problem. If a canvas is connected with a [PRODUCTNAME] window, set both sizes to a valid, positive value, otherwise the [PRODUCTNAME] window will not be visible. If you are using a non-product build of OpenOffice.org, you see an assertion failed "small world isn't it". This might change when the sub-classing is done by VCL in the future.

There is still one unresolved problem. The code mentioned above works with Java 1.3, but not for Java 1.4. There, the behavior of windows is changed. Where Java 1.3 sends real resize events from the own `WindowProc`, Java 1.4 does a re-parenting. The canvas window is destroyed and created again. This leads to an empty window with no OpenOffice.org window. This problem is under investigation.

More Remote Problems

There are additional difficulties to window handles and local window handles. Some personal experiences of one of the OpenOffice.org authors are provided:

- Listeners in Java should be implemented in a thread. The problem is that SolarMutex, a mutex semaphore of OpenOffice.org, one-way UNO methods and the global Java GUI thread do not work together.
- The Java applet should release its listeners. If they stay in the containers of OpenOffice.org after the Java process ends, UNO throws a `com.sun.star.lang.DisposedException`, which are not caught correctly. Java does not know destructors, therefore it is a difficult to follow this advice. One possibility is to register a `Thread` object at `java.Runtime` as a `ShutDownHook`. This is called even when CTRL-C is pressed on the command line where you can deregister the listeners. Because listeners are threads, there is some effort.

6.2 Common Application Features

6.2.1 Clipboard

This chapter introduces the usage of the clipboard service com.sun.star.datatransfer.clipboard.SystemClipboard. The clipboard serves as a data exchange mechanism between OpenOffice.org custom components, or between custom components and external applications. It is usually used for copy and paste operations.



Note: The architecture of the OpenOffice.org clipboard service is strongly conforming to the Java clipboard specification.

Different platforms use different methods for describing data formats available on the clipboard. Under Windows, clipboard formats are identified by unique numbers, for example, under X11, a clipboard format is identified by an ATOM. To have a platform independent mechanism, the OpenOffice.org clipboard supports the concept of DataFlavors. Each instance of a DataFlavor represents the opaque concept of a data format as it would appear on a clipboard. A DataFlavor defined in `com.sun.star.datatransfer.DataFlavor` has three members:

Members of `com.sun.star.datatransfer.DataFlavor`	
`MimeType`	A string that describes the data. This string must conform to Rfc2045 and Rfc2046 with one exception. The quoted parameter may contain spaces. In section *6.2.1 Office Development - Common Application Features - Clipboard - OpenOffice.org Clipboard Data Formats*, a list of common DataFlavors supported by OpenOffice.org is provided.
`HumanPresentableName`	The human presentable name for the data format that this DataFlavor represents.

Members of `com.sun.star.datatransfer.DataFlavor`	
`DataType`	The type of the data. In section *6.2.1 Office Development - Common Application Features - Clipboard - OpenOffice.org Clipboard Data Formats* there is a list of common DataFlavors supported by OpenOffice.org and their corresponding `DataType`.

The carrier of the clipboard data is a transferable object that implements the interface `com.sun.star.datatransfer.XTransferable`. A transferable object offers one or many different DataFlavors.

Using the Clipboard

Pasting Data

The following Java example demonstrates the use of the clipboard service to paste from the clipboard. (OfficeDev/Clipboard/Clipboard.java)

```
import com.sun.star.datatransfer.*;
import com.sun.star.datatransfer.clipboard.*;
import com.sun.star.uno.AnyConverter;
...

// instantiate the clipboard service

Object oClipboard =
         xMultiComponentFactory.createInstanceWithContext(
         "com.sun.star.datatransfer.clipboard.SystemClipboard",
         xComponentContext);

// query for the interface XClipboard

XClipboard xClipboard = (XClipboard)
         UnoRuntime.queryInterface(XClipboard.class, oClipboard);

//-------------------------------------------------
// get a list of formats currently on the clipboard
//-------------------------------------------------

XTransferable xTransferable = xClipboard.getContents();

DataFlavor[] aDflvArr = xTransferable.getTransferDataFlavors();

// print all available formats

System.out.println("Reading the clipboard...");
System.out.println("Available clipboard formats:");

DataFlavor aUniFlv = null;

for (int i=0;i<aDflvArr.length;i++)
{
         System.out.println( "MimeType: " +
                  aDflvArr[i].MimeType +
                  " HumanPresentableName: " +
                  aDflvArr[i].HumanPresentableName );

         // if there is the format unicode text on the clipboard save the
         // corresponding DataFlavor so that we can later output the string

         if (aDflvArr[i].MimeType.equals("text/plain;charset=utf-16"))
         {
                  aUniFlv = aDflvArr[i];
         }
}

System.out.println("");

try
{
         if (aUniFlv != null)
         {
                  System.out.println("Unicode text on the clipboard...");
                  Object aData = xTransferable.getTransferData(aUniFlv);
                  System.out.println(AnyConverter.toString(aData));
         }
```

```
}
catch(UnsupportedFlavorException ex)
{
         System.err.println( "Requested format is not available" );
}

...
```

Copying Data

To copy to the clipboard, implement a transferable object that supports the interface `com.sun.star.datatransfer.XTransferable`. The transferable object offers arbitrary formats described by DataFlavors.

The following Java example demonstrates the implementation of a transferable object. This transferable object contains only one format, unicode text.
(OfficeDev/Clipboard/TextTransferable.java)

```
//---------------------------------------
 // A simple transferable containing only
 // one format, unicode text
 //---------------------------------------

public class TextTransferable implements XTransferable
 {
         public TextTransferable(String aText)
         {
                 text = aText;
         }

         // XTransferable methods
         public Object getTransferData(DataFlavor aFlavor) throws UnsupportedFlavorException
         {
                 if ( !aFlavor.MimeType.equalsIgnoreCase( UNICODE_CONTENT_TYPE ) )
                         throw new UnsupportedFlavorException();
                 return text;
         }
         public DataFlavor[] getTransferDataFlavors()
         {
                 DataFlavor[] adf = new DataFlavor[1];
                 DataFlavor uniflv = new DataFlavor(
                         UNICODE_CONTENT_TYPE,
                         "Unicode Text",
                         new Type(String.class) );
                 adf[0] = uniflv;

                 return adf;
         }
         public boolean isDataFlavorSupported(DataFlavor aFlavor)
         {
                 return aFlavor.MimeType.equalsIgnoreCase(UNICODE_CONTENT_TYPE);
         }

         // members
         private final String text;
         private final String UNICODE_CONTENT_TYPE = "text/plain;charset=utf-16";
 }
```

Everyone providing data to the clipboard becomes a clipboard owner. A clipboard owner is an object that implements the interface `com.sun.star.datatransfer.clipboard.XClipboardOwner`. If the current clipboard owner loses ownership of the clipboard, it receives a notification from the clipboard service. The clipboard owner can use this notification to destroy the transferable object that was formerly on the clipboard. If the transferable object is a self-destroying object, destroying clears all references to the object. If the clipboard service is the last client, clearing the reference to the transferable object leads to destruction.

All data types except for text have to be transferred as byte array. The next example shows this for a bitmap.

```
public class BmpTransferable implements XTransferable
{
    public BmpTransferable(byte[] aBitmap)
    {
        mBitmapData = aBitmap;
    }
```

```
    // XTransferable methods
    public Object getTransferData(DataFlavor aFlavor) throws UnsupportedFlavorException
    {
        if ( !aFlavor.MimeType.equalsIgnoreCase(BITMAP_CONTENT_TYPE ) )
            throw new UnsupportedFlavorException();

        return mBitmapData;
    }
    public DataFlavor[] getTransferDataFlavors()
    {
        DataFlavor[] adf = new DataFlavor[1];
        DataFlavor bmpflv= new DataFlavor(
            BITMAP_CONTENT_TYPE,
            "Bitmap",
            new Type(byte[].class) );
        adf[0] = bmpflv;

        return adf;
    }
    public boolean isDataFlavorSupported(DataFlavor aFlavor)
    {
        return aFlavor.MimeType.equalsIgnoreCase(BITMAP_CONTENT_TYPE);
    }

    // members
    private byte[] mBitmapData;
    private final String BITMAP_CONTENT_TYPE = "application/x-openoffice;windows_formatname="Bitmap"";
}
```

The following Java example shows an implementation of the interface `com.sun.star.datatransfer.clipboard.XClipboardOwner`. (OfficeDev/Clipboard/ClipboardOwner.java)

```
...

//----------------------------------------
// A simple clipboard owner implementation
//----------------------------------------

public class ClipboardOwner implements XClipboardOwner
{
         public void lostOwnership(
                 XClipboard xClipboard,
                 XTransferable xTransferable )
         {
                 System.out.println("");
                 System.out.println( "Lost clipboard ownership..." );
                 System.out.println("");

                 isowner = false;
         }

         public boolean isClipboardOwner()
         {
                 return isowner;
         }

         private boolean isowner = true;
}

...
```

The last two samples combined show how it is possible to copy data to the clipboard as demonstrated in the following Java example. (OfficeDev/Clipboard/Clipboard.java)

```
import com.sun.star.datatransfer.*;
 import com.sun.star.datatransfer.clipboard.*;
 import com.sun.star.uno.AnyConverter;
...

// instantiate the clipboard service

Object oClipboard =
         xMultiComponentFactory.createInstanceWithContext(
         "com.sun.star.datatransfer.clipboard.SystemClipboard",
         xComponentContext);

// query for the interface XClipboard

XClipboard xClipboard = (Xclipboard)UnoRuntime.queryInterface(XClipboard.class, oClipboard);
//------------------------------------------------
 // becoming a clipboard owner
 //------------------------------------------------
```

```
System.out.println("Becoming a clipboard owner...");
 System.out.println("");
ClipboardOwner aClipOwner = new ClipboardOwner();

 xClipboard.setContents(new TextTransferable("Hello World!"), aClipOwner);
while (aClipOwner.isClipboardOwner())
 {
         System.out.println("Still clipboard owner...");
         Thread.sleep(1000);
 }
...
```

Becoming a Clipboard Viewer

It is useful to listen to clipboard changes. User interface controls may change their visible appearance depending on the current clipboard content. To avoid polling on the clipboard, the clipboard service supports an asynchronous notification mechanism. Every client that needs notification about clipboard changes implements the interface `com.sun.star.datatransfer.clipboard.XClipboardListener` and registers as a clipboard listener.

Implementing the interface `com.sun.star.datatransfer.clipboard.XClipboardListener` is simple as the next Java example demonstrates. (OfficeDev/Clipboard/ClipboardListener.java)

```
//----------------------------
 // A simple clipboard listener
 //----------------------------
public class ClipboardListener implements XClipboardListener
 {
         public void disposing(EventObject event)
         {
         }

         public void changedContents(ClipboardEvent event)
         {
                 System.out.println("");
                 System.out.println("Clipboard content has changed!");
                 System.out.println("");
         }
 }
```

If the interface was implemented by the object, it registers as a clipboard listener. A clipboard listener deregisters if clipboard notifications are no longer necessary. Both aspects are demonstrated in the next example. (OfficeDev/Clipboard/Clipboard.java)

```
// instantiate the clipboard service
Object oClipboard =
         xMultiComponentFactory.createInstanceWithContext(
         "com.sun.star.datatransfer.clipboard.SystemClipboard",
         xComponentContext);
// query for the interface XClipboard
XClipboard xClipboard = (XClipboard)
         UnoRuntime.queryInterface(XClipboard.class, oClipboard);
//-----------------------------------------------------
 // registering as clipboard listener
 //-----------------------------------------------------
XClipboardNotifier xClipNotifier = (XClipboardNotifier)
         UnoRuntime.queryInterface(XClipboardNotifier.class, oClipboard);
ClipboardListener aClipListener= new ClipboardListener();
xClipNotifier.addClipboardListener(aClipListener);
...
//-----------------------------------------------------
 // unregistering as clipboard listener
 //-----------------------------------------------------
xClipNotifier.removeClipboardListener(aClipListener);
...
```

OpenOffice.org Clipboard Data Formats

This section describes common clipboard data formats that OpenOffice.org supports and their corresponding DataType.

As previously mentioned, data formats are described by DataFlavors. The important characteris-

tics of a DataFlavor are the MimeType and DataType. The OpenOffice.org clipboard service uses a standard MimeType for different data formats if there is one registered at Iana. For example, for HTML text, the MimeType `"text/html"` is used, Rich Text uses the MimeType `"text/richtext"`, and text uses `"text/plain"`. If there is no corresponding MimeType registered at Iana, OpenOffice.org defines a private MimeType. Private OpenOffice.org MimeType always has the MimeType `"application/x-openoffice"`. Each private OpenOffice.org MimeType has a parameter "windows_formatname" identifying the clipboard format name used under Windows. The used Windows format names are the format names used with older OpenOffice.org versions. Common Windows format names are "Bitmap", `"GDIMetaFile"`, `"FileName"`, `"FileList"`, and `"DIF"`.
The DataType of a DataFlavor identifies how the data are exchanged. There are only two DataTypes that can be used. The DataType for Unicode text is a string, and in Java, `String.class`, For all other data formats, the DataType is a sequence of bytes in Java `byte[].class`.

The following table lists common data formats, and their corresponding MimeType and DataTypes:

Form	***MimeType***	***DataType (in Java)***	***Description***
Unicode Text	text/plain;charset=utf-16	String.class	Unicode Text
Richtext	text/richtext	byte[].class	Richtext
Bitmap	application/x-openoffice;windows_formatname="Bitmap"	byte[].class	A bitmap in OpenOffice bitmap format.
HTML Text	text/html	byte[].class	HTML Text

6.2.2 Internationalization

The I18N framework provides interfaces to access locale-dependent data (e.g. calendar data, currency) and methods (e.g. collation and transliteration). The I18N framework offers full-featured internationalization functionality that covers a range of geographic locations that include South Asia (China, Japan, and Korea, or CJK), Europe, Middle East (Hebrew, Arabic) and South-East Asia (Thai, Indian). Also, the I18N framework builds on the component model UNO, thus making the addition of new internationalization components easy.

Introduction

The I18N framework contains a lot of data and many interfaces and methods not important to developers of external code using the OpenOffice.org API, but only for developers of the OpenOffice.org application itself. This chapter is split into two parts, one that gives a short overview on using the API and is restricted to what is useful to external developers, and a second part that focuses on how to implement a new locale supporting the API (Note that this section does not cover how to translate and localize the OpenOffice.org resources).

XLocaleData

The `com.sun.star.i18n.XLocaleData` interface provides access to locale-specific information, such as decimal separators, group (thousands) separators, currency information, calendar data, and number format codes. No further functionality is discussed.

XCharacterClassification

The `com.sun.star.i18n.XCharacterClassification` interface is used to determine the Unicode type of a character (such as uppercase, lowercase, letter, digit, punctuation) or the script type. It also provides methods to perform simple uppercase to lowercase and lowercase to upper case conversions that are locale-dependent but do not need real transliteration. An example of locale-dependent case conversion is the Turkish lowercase *i* to uppercase *I-dot* and lowercase *i-dotless* to uppercase *I* conversion, as opposed to the western lowercase *i* to uppercase *I* conversion.



There was a bug in OpenOffice.org 1.0.2 that prevents this special example of Turkish case conversion to work properly. The issue is resolved for OpenOffice.org 1.1.

Another provided functionality is parsing methods to isolate and determine identifiers, numbers, and quoted strings in a given string. See the description of `com.sun.star.i18n.XCharacterClassification` methods `parseAnyToken()` and `parsePredefinedToken()`. The parser uses `com.sun.star.i18n.XLocaleData` to obtain the locale-dependent decimal and group separators.

XCalendar

The `com.sun.star.i18n.XCalendar` interface enables the application to use any calendar available for a given locale, not being restricted to the Gregorian calendar. You may query the interface for the available calendars for a given locale with method `getAllCalendars()` and load one of the available calendars using method `loadCalendar()`, or you may use the default calendar loaded with method `loadDefaultCalendar()`. Normally, a Gregorian calendar is available with the name `"gregorian"` in the `Name` field of `com.sun.star.i18n.Calendar` even if the default calendar is not a Gregorian calendar, but this is not mandatory. Available calendars are obtained through the `com.sun.star.i18n.XLocaleData` interface.



You *must* initially load a calendar before using any of the interface methods that perform calendar calculations.

XExtendedCalendar

The `com.sun.star.i18n.XExtendedCalendar` interface was introduced with OpenOffice.org 1.1 and provides additional functionality to display locale and calendar dependent calendar values. This interface is derived from `com.sun.star.i18n.XCalendar`. The interface provides a method to obtain display strings of date parts for specific calendars of a specific locale.

XNumberFormatCode

The `com.sun.star.i18n.XNumberFormatCode` interface provides access to predefined number format codes for a given locale, which in turn are obtained through the `com.sun.star.i18n.XLocaleData` interface. Normally you do not need to bother with it because the application's number formatter *6.2.5 Office Development - Common Application Features - Number*

Formats manages the codes. It just might serve to get the available codes and determine default format codes of a specific category.

XNativeNumberSupplier

The `com.sun.star.i18n.XNativeNumberSupplier` interface was introduced with OpenOffice.org 1.1 and provides functionality to convert between ASCII Arabic digits/numeric strings and native numeral strings, such as Korean number symbols.

XCollator

The `com.sun.star.i18n.XCollator` interface provides locale-dependent collation algorithms for sorting purposes. There is at least one collator algorithm available per locale, though there may be more than one, for example `dictionary` and `telephone` algorithms, or `stroke`, `radical`, `pinyin` in Chinese locales. There is always one default algorithm for each locale that may be loaded using method `loadDefaultCollator()`, and all available algorithms may be queried with method `listCollatorAlgorithms()` of those a selected algorithm may be loaded using `loadCollatorAlgorithm()`. The available collator implementations and options are obtained through the `com.sun.star.i18n.XLocaleData` interface.



You *must* initially load an algorithm prior to using any of the `compare...()` methods, otherwise the result will be 0 indicating any comparison being equal.



Since collation may be a very time consuming procedure, use it only for user-visible data, for example for sorted lists. If, for example, you only need a case insensitive comparison without displaying the results to the user, use the `com.sun.star.i18n.XTransliteration` interface instead.

XTransliteration

The `com.sun.star.i18n.XTransliteration` interface provides methods to perform locale-dependent character conversions, such as case conversions, conversions between Hiragana and Katakana, and Half-width and Full-width. Transliteration is also used by the collators if, for example, a case insensitive sort is to be performed. The available transliteration implementations are obtained through the `com.sun.star.i18n.XLocaleData` interface.



You *must* initially load a transliteration module prior to using any of the transliterating or comparing methods, otherwise the result is unpredictable.



If you only need to determine if two strings are equal for a specific transliteration (for example a case insensitive comparison) use the `equals()` method instead of the `compare...()` methods, it may have a faster implementation.

XBreakIterator

The `com.sun.star.i18n.XBreakIterator` interface may be used to traverse the text in character mode or word mode, to jump to the beginning or to the end of a sentence, to find the beginning or the end of a given script type, and, as the name suggests, to determine a line break position, optionally using a `com.sun.star.linguistic2.XHyphenator`. The service implementation obtains lists of forbidden characters (characters that are not allowed at the beginning or the end of a line in certain locales) through the `com.sun.star.i18n.XLocaleData` interface. The `XBreakIterator` interface also offers methods to determine the script type of a character or to find the beginning or end of a script type along a sequence of characters.

XIndexEntrySupplier

The `com.sun.star.i18n.XIndexEntrySupplier` interface may be used to obtain information on index entries to generate a "table of alphabetical index" for a given locale. Since not all languages are alphabetical in the western sense (for example, CJK languages), different methods are needed.

XExtendedIndexEntrySupplier

The `com.sun.star.i18n.XExtendedIndexEntrySupplier` interface was introduced with OpenOffice.org 1.1 and provides additional functionality to generate index entries for languages that need phonetically sorted indexes, such as Japanese. The interface is derived from `com.sun.star.i18n.XIndexEntrySupplier`.

XInputSequenceChecker

The `com.sun.star.i18n.XInputSequenceChecker` interface was introduced with OpenOffice.org 1.1 and provides input sequence checking for Thai and Hindi.

Implementing a New Locale



The procedures, directory layout, and file contents described here reflect the structure of the i18npool module as of OpenOffice.org version 1.1, and *not* the i18n module for OpenOffice.org 1.0.2.

XLocaleData

One of the most important tasks in implementing a new locale is to define all the locale data to be used, listed in the following table as types returned by the `com.sun.star.i18n.XLocaleData` interface methods:

Type	**Count**
com.sun.star.i18n.LanguageCountryInfo	exactly 1
com.sun.star.i18n.LocaleDataItem	exactly 1
sequence<com.sun.star.i18n.Calendar>	1 or more
sequence<com.sun.star.i18n.Currency>	1 or more
sequence<com.sun.star.i18n.FormatElement>	at least all com.sun.star.i18n.NumberFormatIndex format codes (see below)
sequence<com.sun.star.i18n.Implementation> collator implementations	0 or more, if none specified the ICU collator will be called for the language given in <LanguageCountryInfo>
sequence<string> search options (transliteration modules)	0 or more
sequence<string> collation options (transliteration modules)	0 or more
sequence<string> names of supported transliterations (transliteration modules)	0 or more
com.sun.star.i18n.ForbiddenCharacters	exactly 1, though may have empty elements
sequence<string> reserved words	all words of com.sun.star.i18n.reservedWords

Type	Count
sequence<com.sun.star.beans.PropertyValues> numbering levels (no public XLocaleData API method available, used by and accessible through com.sun.star.text.XDefaultNumberingProvider method getDefaultContinuousNumberingLevels() implemented in i18npool)	exactly 8 <NumberingLevel> entities
sequence<com.sun.star.container.XIndexAccess> outline styles (no public XLocaleData API method available, used by and accessible through com.sun.star.text.XDefaultNumberingProvider method getDefaultOutlineNumberings() implemented in i18npool)	exactly 8 <OutlineStyle> entities consisting of 5 <OutlineNumberingLevel> entities each

Locale data is defined in an XML file. It is translated into a C++ source file during the build process, which is compiled and linked together with other compiled locale data files into shared libraries. The contents of the XML file, their elements, and how they are to be defined are described in *i18npool/source/localedata/data/locale.dtd*. The latest revision available for a specific CVS branch of that file provides up-to-date information about the definitions, as well as additional information.

If the language-country combination is *not* already listed in *tools/inc/lang.hxx* and *tools/source/intntl/isolang.cxx* and *svx/source/dialog/langtab.src,* OpenOffice.org is probably not prepared to deal with your specific locale. For assistance, you can consult *http://l10n.openoffice.org/adding_language.html#step1* (Add the New Language to the Resource System) and join the *dev@l10n.openoffice.org* mailing list (see also *http://l10n.openoffice.org/servlets/ProjectMailingListList*).

In order to conform with the available build infrastructure, the name of your locale data file should follow the conventions used in the *i18npool/source/localedata/data* directory: *<language>_<country>.xml*, where *language* is a lowercase, two letter ISO-639 code, and *country* is an uppercase two letter ISO-3166 code. Start by copying the *en_US.xml* file to your *<language>_<country>.xml* file and adopt the entries to suit your needs. Add the corresponding **.cxx* and **.obj* target file name to the *i18npool/source/localedata/data/makefile.mk*. Note that there is an explicit rule defined, so that you do not need to add the **.xml* file name anywhere. You must also add the locale to the `aDllsTable` structure located in *i18npool/source/localedata/data/localedata.cxx*. Make sure to specify the correct library name, since it must correspond to the library name used in the makefile. Finally, the public symbols to be exported must be added to the linker map file corresponding to the library. You can use the *i18npool/source/localedata/data/linkermapfile-check.awk* script to assist you. Instructions for how to use the script are located the header comments of the file.

<LC_FORMAT><FormatElement>

To be able to load documents of versions up to and including StarOffice 5.2 (old binary file format), each locale must define all number formats mentioned in `com.sun.star.i18n.NumberFormatIndex` and assign the proper `formatindex="..."` attribute.

Failing to do so may result in data not properly displayed or not displayed at all if a built-in "System" or "Default" format code was used (as generally done by the average user) and the document is loaded under a locale not having those formats defined. Since old versions did merge some format information of the [Windows] Regional Settings, it might be necessary to define some duplicated codes to fill all positions. To verify that all necessary elements are defined, use a non-product build of OpenOffice.org and open a number formatting dialog, and select your locale from the **Language** list box. An assertion message box appears if there are

any missing elements. The errors are only shown the very first time the locale is selected in a given document.

<LC_FORMAT><FormatElement><FormatCode>

In general, definition of number format codes follows the user visible rules, apart from that any non-ASCII character must be entered using UTF-8 encoding. For a detailed description of codes and a list of possible keywords please consult the OpenOffice.org English online help on section "number format codes".

Be sure to use the separators you declared in the `<LC_CTYPE>` section in the number format codes, for example `<DecimalSeparator>`, `<ThousandSeparator>`, otherwise the number formatter generates incorrect formats.

Verify the defined codes again by using the number formatter dialog of a non-product OpenOffice.org build. If anything is incorrect, an assertion message box appears containing information about the error.

The format indices 1..49 are reserved and, for backward compatibility, *must* be used as stated in *offapi/com/sun/star/i18n/NumberFormatIndex.idl*. Note that 48 and 49 are used internally and must not be used in locale data XML files. All other formats must be present.

<FormatCode usage="DATE"> and <FormatCode usage="DATE_TIME">

Characters of date and time keywords, such as YYYY for year, had previously been localized for a few locales (for example, JJJJ in German). The new I18N framework no longer follows that approach, because it may lead to ambiguous and case insensitive character combinations that cannot be resolved at runtime. Localized keyword support is only given for some old locales, other locales must define their codes using English notation.

The table below shows the localized keyword codes:

	DayOfWeek	**Era**	**Year**	**Month**	**Day**	**Hour**
English (and all other locales not mentioned)	A	G	Y	M	D	H
de_AT, de_CH, de_DE, de_LI, de_LU			J		T	
nl_BE, nl_NL			J			U
fr_BE, fr_CA, fr_CH, fr_FR, fr_LU, fr_MC	O		A		J	
it_CH, it_IT	O	X	A		G	
pt_BR, pt_PT	O		A			
es_AR, es_BO, es_CL, es_CO, es_CR, es_DO, es_EC, es_ES, es_GT, es_HN, es_MX, es_NI, es_PA, es_PE, es_PR, es_PY, es_SV, es_UY, es_VE	O		A			
da_DK						T
nb_NO, nn_NO, no_NO						T
sv_FI, sv_SE						T
fi_FI			V	K	P	T

<FormatCode usage="DATE" formatindex="21"> and
<FormatCode usage="DATE_TIME" formatindex="47">

The `formatindex="21"` com.sun.star.i18n.NumberFormatIndex `DATE_SYS_DDMMYYYY` format code is used to edit date formatted data. It represents a date using the most detailed information available, for example, a 4-digit year and instead of a 2-digit year. The YMD default order

(how a date is assembled) is determined from the order encountered in this format. Similarly, the `formatindex="47"` `com.sun.star.i18n.NumberFormatIndex DATETIME_SYS_DDMMYYYY_HHMMSS` format code is used to edit date-time data. Both format codes must display data in a way that is parable by the application, in order to be able to reassemble edited data. This generally means using only YYYY,MM,DD,HH,MM,SS keywords and `<DateSeparator>` and `<TimeSeparator>`.

<FormatCode usage="CURRENCY">

The [$xxx-yyy] notation is needed for compatibility reasons. The `xxx` part denotes the currency symbol, and the `yyy` part specifies the locale identifier in Microsoft Language ID hexadecimal notation. For example, having "409" as the locale identifier (English-US) and "$" as the currency symbol results in [$$-409]. A list of available Language IDs known to the OpenOffice.org application can be found at project util module tools in file *tools/inc/lang.hxx*. Format indices 12, 13, 14, 15, 17 with [$xxx-yyy] notation must use the xxx currency symbol that has the attribute usedInCompatibleFormatCodes="true" (see element `<LC_CURRENCY>` in the *locale.dtd* file).

XCalendar

The interface `com.sun.star.i18n.XCalendar` provides a general calendar service. All calendar implementations are managed by a class `CalendarImpl`, the front-end, which dynamically calls a language-specific implementation.

`Calendar_gregorian` is a wrapper to ICU's Calendar class.

If you need to implement a locale-specific calendar, you can choose to either derive your class from `Calendar_gregorian` or to write your own class.

There are three steps needed to create a locale-specific calendar:

1. Name your calendar `<name>` (for example, 'gengou' for Japanese Calendar) and add it to the locale data XML file with proper day/month/era names.
2. Derive a class either from `Calendar_gregorian` or `XCalendar`, name it as `Calendar_<name>`, which will be loaded by `CalendarImpl` when the calendar is specified.
3. Add your new calendar as a service in *i18npool/source/registerservices/registerservices.cxx*.

If you plan to derive from the Gregorian calendar, you need to know the mapping between your new calendar and the Gregorian calendar. For example, the Japanese Emperor Era calendar has a starting year offset to Gregorian calendar for each era. You will need to override the method `Calendar_gregorian::mapToGregorian()` and `Calendar_gregorian::mapFromGregorian()` to map the Era/Year/Month/Day between the Gregorian calendar and the calendar for your language.

XCharacterClassification

The interface `com.sun.star.i18n.XCharacterClassification` provides `toUpper()`, `toLower()`, `toTitle()` and methods to get various character attributes defined by Unicode. These functions are implemented by the `cclass_unicode` class. If you need language specific requirements for these functions, you can derive a language specific class `cclass_<locale_name>` from `cclass_unicode` and overwrite the methods. In most cases, the attributes are well defined by Unicode, so you do not need to create your own class.

The class also provides a generic parser. If a particular language needs special number parsing, detected non-ASCII numbers are fed to the `com.sun.star.i18n.NativeNumberSupplier` service

to obtain the ASCII representation, which in turn is interpreted and converted to a double precision floating point value.

A manager class `CharacterClassificationImpl` will handle the loading of language specific implementations of `CharacterClassification` on the fly. If no implementation is provided, the implementation defaults to class `cclass_unicode`.

XBreakIterator

The interface `com.sun.star.i18n.XBreakIterator` provides support for Character(Cell)/Word/Sentence/Line-break services. For example, `BreakIterator` provides the APIs to iterate a string by character, word, line and sentence. The interface is used by the Output layer for the following operations:

- Cursor positioning and selection: Since a character or cell can take more than one code point, cursor movement cannot be done by simply incrementing or decrementing the index.
- Complex Text Layout Languages (CTL): In CTL languages (such as Thai, Hebrew, Arabic and Indian), multiple characters can combine to form a display cell. Cursor movement must traverse a display cell instead of a single character.

Line breaking must be highly configurable in desktop publishing applications. The line breaking algorithm should be able to find a line break with or without a hyphenator. Additionally, it should be able to parse special characters that are illegal if they occur at the end or beginning of a line.

Both requirements are locale-sensitive.

The `BreakIterator` components are managed by the class `BreakIteratorImpl`, which will load the language-specific component in service name `BreakIterator_<language>` dynamically.

The base break iterator class `BreakIterator_Unicode` is a wrapper to the ICU `BreakIterator` class. While this class meets the requirements for western languages, it does not meet the requirements for other languages, such as those of South Asia (CJK) and South East Asia (Indian, Thai, Arabic), where enhanced functionality is required, as described previously.

Thus the current `BreakIterator` base class has two derived classes, `BreakIterator_CJK` and `BreakIterator_CTL`. `BreakIterator_CJK` provides a dictionary based word break for Chinese and Japanese, and a forbidden rule driven line break for Chinese, Japanese and Korean. `BreakIterator_CTL` provides a more specific definition of character/cell/cluster grouping for languages like Thai and Arabic.

Use the following steps to create a language-specific `BreakIterator` service:

1. Derive a class either from `BreakIterator_CJK` or `BreakIterator_CTL`, name it as `BreakIterator_<language>`.
2. Add new service in `registerservices.cxx`

There are three methods for word breaking: `nextWord()`, `previousWord()`, `getWordBoundary()`. You can overwrite them with your own language rules.

`BreakIterator_CJK` provides input string caching and dictionary searching for longest matching. You can provide a sorted dictionary (the encoding must be UTF-8) by creating the following file: *i18npool/source/breakiterator/data/<language>.dict*.

The utility `gendict` will convert the file to C code, which will be compiled into a shared library for dynamic loading.

All dictionary searching and loading is performed in the `xdictionary` class. The only thing you need to do is to derive your class from `BreakIterator_CJK` and create an instance of the `xdictionary` with the language name and pass it to the parent class.

XCollator

The interface `com.sun.star.i18n.XCollator` must be used to provide text collation for the new locale. There are two types of collations, *single level* and *multiple level* collation.

Most European and English locales need multiple level collation. OpenOffice.org uses the ICU collator to cover these needs.

Most CJK languages only require single level collation. There is a two step lookup table that performs the collation for these languages. If you have a new language or algorithm in this category, you can derive a new service from `Collator_CJK` and provide index and weight tables. Here is a sample implementation:

```
#include <collator_CJK.hxx>
static sal_uInt16 index[] = {
...
};

static sal_uInt16 weight[] = {
...
};

sal_Int32 SAL_CALL Collator_zh_CN_pinyin::compareSubstring(
        const ::rtl::OUString& str1, sal_Int32 off1, sal_Int32 len1,
        const ::rtl::OUString& str2, sal_Int32 off2, sal_Int32 len2)
        throw (::com::sun::star::uno::RuntimeException)
{
    return compare(str1, off1, len1, str2, off2, len2, index, weight);
}

sal_Int32 SAL_CALL Collator_zh_CN_pinyin::compareString(
        const ::rtl::OUString& str1,
        const ::rtl::OUString& str2)
        throw (::com::sun::star::uno::RuntimeException)
{
    return compare(str1, 0, str1.getLength(), str2, 0, str2.getLength(),
        index, weight);
}
```

Front end implementation `Collator` will load and cache the language-specific service on the name `Collator_<locale>` dynamically.

The steps to add new services:

1. Derive the new service from the above class
2. Provide the index and weight tables
3. Register the new service in *registerservices.cxx*
4. Add the new service in the collation section in the locale data file.

XTransliteration

The interface `com.sun.star.i18n.XTransliteration` can be used for string conversion. The front end implementation `TransliterationImpl` will load and cache specific transliteration services by a predefined enum in `com.sun.star.i18n.TransliterationModules` or `com.sun.star.i18n.TransliterationModulesNew`, or dynamically by implementation name.

Transliterations have been defined in three categories: `Ignore`, `OneToOne` and `Numeric`. All of them are derived from `transliteration_commonclass`.

`Ignore` services are for ignore case, half/full width, and Katakana/Hiragana. You can derive your new service from it, and overwrite folding/transliteration methods.

`OneToOne` services are for one to one mapping, such as converting lowercase to uppercase. The class provides two more services, to take a mapping table or mapping function to do folding and transliteration. You can derive a class from it and provide a table or function for the parent class to do the transliteration.

`Numeric` services are used to convert a number to a number string in specific languages. It can be used to format Date string and other types of strings.

To add a new transliteration

1. Derive a new class from the three classes previously mentioned.
2. Overwrite folding/transliteration methods or provide a table for the parent to perform the transliteration.
3. Register the new service in *registerservices.cxx*
4. Add the new service in the transliteration section in the locale data file

XNativeNumberSupplier

The interface `com.sun.star.i18n.XNativeNumberSupplier` provides the functionality to convert between ASCII Arabic digit numbers and locale-dependent numeral representations. It performs the conversion by implementing special transliteration services. The interface also provides a mechanism to generate attributes to be stored in the XML file format (see the XML file format documentation, section "Common Data Style Attributes", "number:transliteration-..."), as well as a conversion of those XML attributes needed to map back to a specific representation style. If you add a number transliteration for a specific locale and reuse one of the `com.sun.star.i18n.NativeNumberMode` constants, please add the description to `com.sun.star.i18n.NativeNumberMode` if your changes are to be added back to the OpenOffice.org code repository.

XIndexEntrySupplier

The interface `com.sun.star.i18n.XIndexEntrySupplier` can be used to provide the functionality to generate index pages. The main method of this interface is `getIndexCharacter()`. Front end implementation `IndexEntrySupplier` will dynamically load and cache language specific service based on the name `IndexEntrySupplier_<locale>`.

Languages to be indexed have been divided into two sets. The first set contains Latin1 languages, which can be covered by 256 Unicode code points. A one step lookup table is used to generate index characters. An alphabetic and numeric table has been generated, which covers most Latin1 languages. But if you need another algorithm or have a conflict with the table, you can create your own table and derive a new class from `IndexEntrySupplier_Euro`. Here is a sample implementation:

```
#include <sal/types.h>
#include <indexentrysupplier_euro.hxx>
#include <indexdata_alphanumeric.h>

OUString SAL_CALL i18n::IndexEntrySupplier_alphanumeric::getIndexCharacter(
        const OUString& rIndexEntry,
        const lang::Locale& rLocale, const OUString& rSortAlgorithm )
        throw (uno::RuntimeException)
{
    return getIndexString(rIndexEntry, idxStr);
}
```

where `idxStr` is the table.

For the languages that could not be covered in the first set, such as CJK, a two step lookup table is used. Here is a sample implementation:

```
#include <indexentrysupplier_cjk.hxx>
#include <indexdata_zh_pinyin.h>

OUString SAL_CALL i18n::IndexEntrySupplier_zh_pinyin::getIndexCharacter(
        const OUString& rIndexEntry,
        const lang::Locale& rLocale, const OUString& rSortAlgorithm )
        throw (uno::RuntimeException)
{
    return getIndexString(rIndexEntry, idxStr, idx1, idx2);
}
```

where `idx1` and `idx2` are two step tables and `idxStr` contains all the index keys that will be returned. If you have a new language or algorithm, you can derive a new service from `IndexEntrySupplier_CJK` and provide tables for the parent class to generate the index.

Note that the index depends on collation, therefore, each index algorithm should have a collation algorithm to support it.

To add new service:

1. Derive the new service from `IndexEntrySupplier_Euro`.
2. Provide a table for the lookup
3. Register new service in `registerservices.cxx`

A Comment on Search and Replace

Search and replace is also locale-dependent because there may be special search options that are only available for a particular locale. For instance, if the **Asian languages support** is enabled, you'll see an additional option for "Sounds like (Japanese)" in the **Edit - Find & Replace** dialog box. With this option, you can turn on or off certain options specific to Japanese in the search and replace process.

Search and replace relies on the transliteration modules for various search options. The transliteration modules are loaded and the search string is converted before the search process.

6.2.3 Linguistics

The Linguistic API provides a set of UNO services used for spell checking, hyphenation or accessing a thesaurus. Through the Linguistic API, developers add new implementations and integrate them into OpenOffice.org. Users of the Linguistic API call its methods Usually this functionality is used by one or more clients, that is, applications or components, to process documents , such as text documents or spreadsheets.

Services Overview

The services provided by the Linguistic API are:

- `com.sun.star.linguistic2.LinguServiceManager`
- `com.sun.star.linguistic2.DictionaryList`
- `com.sun.star.linguistic2.LinguProperties`

Also there is at least one or more implementation for each of the following services:

- com.sun.star.linguistic2.SpellChecker
- com.sun.star.linguistic2.Hyphenator
- com.sun.star.linguistic2.Thesaurus

The service implementations for spell checker, thesaurus and hyphenator supply the respective functionality. Each of the implementations support a different set of languages. Refer to com.sun.star.linguistic2.XSupportedLocales.

For example, there could be two implementations for a spell checker, usually from different supporting parties: the first supporting English, French and German, and the second supporting Russian and English. Similar settings occur for the hyphenator and thesaurus.

It is not convenient for each application or component to know all these implementations and to choose the appropriate implementation for the specific purpose and language, therefore a mediating instance is required.

This instance is the LinguServiceManager. Spell checking, hyphenation and thesaurus functionality is accessed from a client by using the respective interfaces from the LinguServiceManager.

The LinguServiceManager dispatches the interface calls from the client to a specific service implementation,if any, of the respective type that supports the required language.
For example, if the client requires spell checking of a French word, the first spell checker implementations from those mentioned above are called.

If there is more than one spell checker available for one language, as in the above example for the English language, the LinguServiceManager starts with the first one that was supplied in the setConfiguredServices() method of its interface. The thesaurus behaves in a similar manner. For more details, refer to the interface description com.sun.star.linguistic2.XLinguServiceManager.

The LinguProperties service provides, among others, properties that are required by the spell checker, hyphenator and thesaurus that are modified by the client. Refer to the com.sun.star.linguistic2.LinguProperties.

The DictionaryList (see com.sun.star.linguistic2.DictionaryList) provides a set of user defined or predefined dictionaries for languages that are activated and deactivated. If they are active, they are used by the spell checker and hyphenator. These are used by the user to override results from the spell checker and hyphenator implementations, thus allowing the user to customize spell checking and hyphenation.

In the code snippets and examples in the following chapters, we will use the following members and interfaces: (OfficeDev/Linguistic/LinguisticExamples.java)

```
// used interfaces
import com.sun.star.lang.XMultiServiceFactory;
import com.sun.star.linguistic2.XLinguServiceManager;
import com.sun.star.linguistic2.XSpellChecker;
import com.sun.star.linguistic2.XHyphenator;
import com.sun.star.linguistic2.XThesaurus;
import com.sun.star.linguistic2.XSpellAlternatives;
import com.sun.star.linguistic2.XHyphenatedWord;
import com.sun.star.linguistic2.XPossibleHyphens;
import com.sun.star.linguistic2.XMeaning;
import com.sun.star.linguistic2.XSearchableDictionaryList;
import com.sun.star.linguistic2.XLinguServiceEventListener;
import com.sun.star.linguistic2.LinguServiceEvent;
import com.sun.star.beans.XPropertySet;
import com.sun.star.beans.PropertyValue;
import com.sun.star.uno.XComponentContext;
import com.sun.star.uno.XNamingService;
import com.sun.star.lang.XMultiComponentFactory;
import com.sun.star.lang.EventObject;
import com.sun.star.lang.Locale;
import com.sun.star.bridge.XUnoUrlResolver;
import com.sun.star.uno.UnoRuntime;
import com.sun.star.uno.Any;
import com.sun.star.lang.XComponent;
```

```
//
// members for commonly used interfaces
//

// The MultiServiceFactory interface of the Office
protected XMultiServiceFactory mxFactory = null;

// The LinguServiceManager interface
protected XLinguServiceManager mxLinguSvcMgr = null;

// The SpellChecker interface
protected XSpellChecker mxSpell = null;

// The Hyphenator interface
protected XHyphenator mxHyph = null;

// The Thesaurus interface
protected XThesaurus mxThes = null;

// The DictionaryList interface
protected XSearchableDictionaryList mxDicList = null;

// The LinguProperties interface
protected XPropertySet mxLinguProps = null;
```

To establish a connection to the office and have our `mxFactory` object initialized with its `XMultiServiceFactory`, the following code is used: (OfficeDev/Linguistic/LinguisticExamples.java)

```
public void Connect( String sConnection )
    throws com.sun.star.uno.Exception,
    com.sun.star.uno.RuntimeException,
    Exception
{
    XComponentContext xContext =
        com.sun.star.comp.helper.Bootstrap.createInitialComponentContext( null );
    XMultiComponentFactory xLocalServiceManager = xContext.getServiceManager();

    Object  xUrlResolver  = xLocalServiceManager.createInstanceWithContext(
        "com.sun.star.bridge.UnoUrlResolver", xContext );
    XUnoUrlResolver urlResolver = (XUnoUrlResolver)UnoRuntime.queryInterface(
        XUnoUrlResolver.class, xUrlResolver );
    Object rInitialObject = urlResolver.resolve( "uno:" + sConnection +
        ";urp;StarOffice.NamingService" );
    XNamingService rName = (XNamingService)UnoRuntime.queryInterface(XNamingService.class,
        rInitialObject );
    if( rName != null )
    {
        Object rXsmgr = rName.getRegisteredObject( "StarOffice.ServiceManager" );
        mxFactory = (XMultiServiceFactory)
            UnoRuntime.queryInterface( XMultiServiceFactory.class, rXsmgr );
    }
}
```

And the `LinguServiceManager` object `mxLinguSvcMgr` is initialized like similar to the following snippet: (OfficeDev/Linguistic/LinguisticExamples.java)

```
/** Get the LinguServiceManager to be used. For example to access spell checker,
    thesaurus and hyphenator, also the component may choose to register itself
    as listener to it in order to get notified of relevant events. */
public boolean GetLinguSvcMgr()
    throws com.sun.star.uno.Exception
{
    if (mxFactory != null) {
        Object aObj = mxFactory.createInstance(
            "com.sun.star.linguistic2.LinguServiceManager" );
        mxLinguSvcMgr = (XLinguServiceManager)
                UnoRuntime.queryInterface(XLinguServiceManager.class, aObj);
    }
    return mxLinguSvcMgr != null;
}
```

The empty list of temporary property values used for the current function call only and the language used may look like the following:

```
// list of property values to used in function calls below.
// Only properties with values different from the (default) values
// in the LinguProperties property set need to be supllied.
// Thus we may stay with an empty list in order to use the ones
// form the property set.
PropertyValue[] aEmptyProps = new PropertyValue[0];

// use american english as language
```

```
Locale aLocale = new Locale("en","US","");
```

Using temporary property values:

To change a value for the example `IsGermanPreReform` to a different value for one or a limited number of calls without modifying the default values, provide this value as a member of the last function argument used in the examples below before calling the respective functions.

```
// another list of property values to used in function calls below.
// Only properties with values different from the (default) values
// in the LinguProperties property set need to be supllied.
PropertyValue[] aProps = new PropertyValue[1];
aProps[0] = new PropertyValue();
aProps[0].Name  = "IsGermanPreReform";
aProps[0].Value = new Boolean( true );
```

Replace the `aEmptyProps` argument in the function calls with `aProps` to override the value of `IsGermanPreReform` from the `LinguProperties`. Other properties are overridden by adding them to the `aProps` object.

Using Spellchecker

The interface used for spell checking is `com.sun.star.linguistic2.XSpellChecker`. Accessing the spell checker through the `LinguServiceManager` and initializing the `mxSpell` object is done by: (OfficeDev/Linguistic/LinguisticExamples.java)

```
/** Get the SpellChecker to be used.
*/
public boolean GetSpell()
    throws com.sun.star.uno.Exception,
    com.sun.star.uno.RuntimeException
{
    if (mxLinguSvcMgr != null)
        mxSpell = mxLinguSvcMgr.getSpellChecker();
    return mxSpell != null;
}
```

Relevant properties

The properties of the `LinguProperties` service evaluated by the spell checker are:

Spell-checking Properties of `com.sun.star.linguistic2.LinguProperties`	**Description**
`IsIgnoreControlCharacters`	Defines if control characters should be ignored or not.
`IsUseDictionaryList`	Defines if the dictionary-list should be used or not.
`IsGermanPreReform`	Defines if the new German spelling rules should be used for German language text or not.
`IsSpellUpperCase`	Defines if words with only uppercase letters should be subject to spellchecking or not.
`IsSpellWithDigits`	Defines if words containing digits or numbers should be subject to spellchecking or not.
`IsSpellCapitalization`	dDefines if the captitalization of words should be checked or not.

Changing the values of these properties in the `LinguProperties` affect all subsequent calls to the spell checker. Instantiate a `com.sun.star.linguistic2.LinguProperties` instance and change it by calling `com.sun.star.beans.XPropertySet:setPropertyValue()`. The changes affect the whole office unless another modifies the properties again. This is done implicitly when changing the linguistic settings through **Tools - Options - Language Settings - Writing Aids**.

The following example shows verifying single words:
(OfficeDev/Linguistic/LinguisticExamples.java)

```
// test with correct word
```

```
String aWord = "horseback";
boolean bIsCorrect = mxSpell.isValid( aWord, aLocale, aEmptyProps );
System.out.println( aWord + ": " +  bIsCorrect );

// test with incorrect word
aWord = "course";
bIsCorrect = mxSpell.isValid( aWord, aLocale , aEmptyProps );
System.out.println( aWord + ": " +  bIsCorrect );
```

Tne following example shows spelling a single word and retrieving possible corrections:

```
aWord = "house";
XSpellAlternatives xAlt = mxSpell.spell( aWord, aLocale, aEmptyProps );
if (xAlt == null)
    System.out.println( aWord + " is correct." );
else
{
    System.out.println( aWord + " is not correct. A list of proposals follows." );
    String[] aAlternatives = xAlt.getAlternatives();
    if (aAlternatives.length == 0)
        System.out.println( "no proposal found." );
    else
    {
        for (int i = 0; i < aAlternatives.length; ++i)
            System.out.println( aAlternatives[i] );
    }
}
```

For a description of the return types interface, refer to `com.sun.star.linguistic2.XSpellAlternatives`.

Using Hyphenator

The interface used for hyphenation is `com.sun.star.linguistic2.XHyphenator`. Accessing the hyphenator through the `LinguServiceManager` and initializing the `mxHyph` object is done by: (OfficeDev/Linguistic/LinguisticExamples.java)

```
/** Get the Hyphenator to be used.
*/
public boolean GetHyph()
    throws com.sun.star.uno.Exception,
    com.sun.star.uno.RuntimeException
{
    if (mxLinguSvcMgr != null)
        mxHyph = mxLinguSvcMgr.getHyphenator();
    return mxHyph != null;
}
```

Relevant properties

The properties of the LinguProperties service evaluated by the hyphenator are:

Hyphenating Properties of `com.sun.star.linguistic2.LinguProperties`	
`IsIgnoreControlCharacters`	Defines if control characters should be ignored or not.
`IsUseDictionaryList`	Defines if the dictionary-list should be used or not.
`IsGermanPreReform`	Defines if the new German spelling rules should be used for German language text or not.
`HyphMinLeading`	The minimum number of characters of a hyphenated word to remain before the hyphenation character.
`HyphMinTrailing`	The minimum number of characters of a hyphenated word to remain after the hyphenation character.
`HyphMinWordLength`	The minimum length of a word to be hyphenated.

Changing the values of these properties in the Lingu-Properties affect all subsequent calls to the hyphenator.

A *valid* hyphenation position is a possible one that meets the restrictions given by the `HyphMin-Leading`, `HyphMinTrailing` and `HyphMinWordLength` values.

For example, if `HyphMinWordLength` is `7`, "remove" does not have a *valid* hyphenation position. Also, this is the case when `HyphMinLeading` is `3` or `HyphMinTrailing` is `5`.

The following example shows a word hypenated: (OfficeDev/Linguistic/LinguisticExamples.java)

```
// maximum number of characters to remain before the hyphen
// character in the resulting word of the hyphenation
short nMaxLeading = 6;

XHyphenatedWord xHyphWord = mxHyph.hyphenate( "horseback", aLocale, nMaxLeading , aEmptyProps );
if (xHyphWord == null)
    System.out.println( "no valid hyphenation position found" );
else
{
    System.out.println( "valid hyphenation pos found at " + xHyphWord.getHyphenationPos()
            + " in " + xHyphWord.getWord() );
    System.out.println( "hyphenation char will be after char " + xHyphWord.getHyphenPos()
            + " in " + xHyphWord.getHyphenatedWord() );
}
```

If the hyphenator implementation is working correctly, it reports a *valid* hyphenation position of 4 that is after the 'horse' part. Experiment with other values for `nMaxLeading` and other words. For example, if you set it to 4, no *valid* hyphenation position is found since there is no hyphenation position in the word 'horseback' before and including the 's'.

For a description of the return types interface, refer to `com.sun.star.linguistic2.XHyphenatedWord`.

The example below shows querying for an alternative spelling. In some languages, for example German in the old (pre-reform) spelling, there are words where the spelling of changes when they are hyphenated at specific positions. To inquire about the existence of alternative spellings, the `queryAlternativeSpelling()` function is used: (OfficeDev/Linguistic/LinguisticExamples.java)

```
//! Note: 'aProps' needs to have set 'IsGermanPreReform' to true!
xHyphWord = mxHyph.queryAlternativeSpelling( "Schiffahrt",
                    new Locale("de","DE",""), (short)4, aProps );
if (xHyphWord == null)
    System.out.println( "no alternative spelling found at specified position." );
else
{
    if (xHyphWord.isAlternativeSpelling())
        System.out.println( "alternative spelling detectetd!" );
    System.out.println( "valid hyphenation pos found at " + xHyphWord.getHyphenationPos()
            + " in " + xHyphWord.getWord() );
    System.out.println( "hyphenation char will be after char " + xHyphWord.getHyphenPos()
            + " in " + xHyphWord.getHyphenatedWord() );
}
```

The return types interface is the same as in the above example (`com.sun.star.linguistic2.XHyphenatedWord`).

The next example demonstrates getting possible hyphenation positions. To determine all possible hyphenation positions in a word, do this: (OfficeDev/Linguistic/LinguisticExamples.java)

```
XPossibleHyphens xPossHyph = mxHyph.createPossibleHyphens( "waterfall", aLocale, aEmptyProps );
if (xPossHyph == null)
    System.out.println( "no hyphenation positions found." );
else
    System.out.println( xPossHyph.getPossibleHyphens() );
```

For a description of the return types interface, refer to `com.sun.star.linguistic2.XPossibleHyphens`.

Using Thesaurus

The interface used for the thesaurus is `com.sun.star.linguistic2.XThesaurus`. Accessing the thesaurus through the `LinguServiceManager` and initializing the `mxThes` object is done by: (OfficeDev/Linguistic/LinguisticExamples.java)

```
/** Get the Thesaurus to be used.
```

```
*/
public boolean GetThes()
    throws com.sun.star.uno.Exception,
    com.sun.star.uno.RuntimeException
{
    if (mxLinguSvcMgr != null)
        mxThes = mxLinguSvcMgr.getThesaurus();
    return mxThes != null;
}
```

The properties of the `LinguProperties` service evaluated by the thesaurus are:

Thesaurus related Properties of `com.sun.star.linguistic2.LinguProperties`	
`IsIgnoreControlCharacters`	Defines if control characters should be ignored or not.
`IsGermanPreReform`	Defines if the new German spelling rules should be used for German language text or not.

Changing the values of these properties in the `LinguProperties` affect all subsequent calls to the thesaurus. The following example about retrieving synonyms shows this: (OfficeDev/Linguistic/LinguisticExamples.java)

```
XMeaning[] xMeanings = mxThes.queryMeanings( "house", aLocale, aEmptyProps );
if (xMeanings == null)
    System.out.println( "nothing found." );
else
{
    for (int i = 0; i < xMeanings.length; ++i)
    {
        System.out.println( "Meaning: " + xMeanings[i].getMeaning() );
        String[] aSynonyms = xMeanings[i].querySynonyms();
        for (int k = 0; k < aSynonyms.length; ++k)
            System.out.println( "    Synonym: " + aSynonyms[k] );
    }
}
```

The reason to subdivide synonyms into different meanings is becausethere are different synonyms for some words that are not even closely related. For example, the word 'house' has the synonyms 'home', 'place', 'dwelling', 'family', 'clan', 'kindred', 'room', 'board', and 'put up'.

The first three in the aboce list have the meaning of 'building where one lives' where the next three mean that of 'a group of people sharing common ancestry' and the last three means that of 'to provide with lodging'. Thus, having meanings is a way to group large sets of synonyms into smaller ones with approximately the same definition.

Events

There are several types of events. For example, all user dictionaries `com.sun.star.linguistic2.XDictionary` report their status changes as events `com.sun.star.linguistic2.DictionaryEvent` to the `DictionaryList`, which collects and transforms their information into `DictionaryList` events `com.sun.star.linguistic2.DictionaryListEvent`, and passes those on to its own listeners.

Thus, it is possible to register to the `DictionaryList` as a listener to be informed about relevant changes in the dictionaries., There is no need to register as a listener for each dictionary.

The spell checker and hyphenator implementations monitor the changes in the `LinguProperties` for changes of their relevant properties. If such a property changes its value, the implementation launches an event `com.sun.star.linguistic2.LinguServiceEvent`that hints to its listeners that spelling or hyphenation should be reevaluated. For this purpose, those implementations support the `com.sun.star.linguistic2.XLinguServiceEventBroadcaster` interface.

The `LinguServiceManager` acts as a listener for `com.sun.star.linguistic2.DictionaryListEvent` and `com.sun.star.linguistic2.LinguServiceEvent` events. The respective interfaces are `com.sun.star.linguistic2.XDictionaryListEventListener]` and

com.sun.star.linguistic2.XLinguServiceEventListener. The events from the DictionaryList are transformed into com.sun.star.linguistic2.LinguServiceEvent events and passed to the listeners of the LinguServiceManager, along with the received events from the spell checkers and hyphenators.

Therefore, a client that wants to be notified when spell checking or hyphenation changes, for example, when it features automatic spell checking or automatic hyphenation, needs to be registered as com.sun.star.linguistic2.XLinguServiceEventListener to the LinguServiceManager only.

Implementing the com.sun.star.linguistic2.XLinguServiceEventListener interface is similar to the following snippet: (OfficeDev/Linguistic/LinguisticExamples.java)

```
/** simple sample implementation of a clients XLinguServiceEventListener
    *  interface implementation
    */
public class Client
        implements XLinguServiceEventListener
{
    public void disposing ( EventObject aEventObj )
    {
        //! any references to the EventObjects source have to be
        //! released here now!

        System.out.println("object listened to will be disposed");
    }

    public void processLinguServiceEvent( LinguServiceEvent aServiceEvent )
    {
        //! do here whatever you think needs to be done depending
        //! on the event recieved (e.g. trigger background spellchecking
        //! or hyphenation again.)

        System.out.println("Listener called");
    }
};
```

After the client has been instantiated, it needs to register as com.sun.star.linguistic2.XLinguServiceEventListener. For the sample client above, this looks like: (OfficeDev/Linguistic/LinguisticExamples.java)

```
XLinguServiceEventListener aClient = new Client();

// now add the client as listener to the service manager to
// get informed when spellchecking or hyphenation may produce
// different results then before.
mxLinguSvcMgr.addLinguServiceManagerListener( aClient );
```

This enables the sample client to receive com.sun.star.linguistic2.LinguServiceEvents and act accordingly. Before the sample client terminates, it has to stop listening for events from the LinguServiceManager:

```
//! remove listener before programm termination.
//! should not be omitted.
mxLinguSvcMgr.removeLinguServiceManagerListener(aClient);
```

In the *LinguisticExamples.java* sample, a property is modified for the listener to be called.

Implementing a Spell Checker

A sample implementation of a spell checker isfound in the (OfficeDev/Linguistic/SampleSpellChecker.java) file from the examples for linguistics.

The spell checker implements the following interfaces:

- com.sun.star.linguistic2.XSpellChecker
- com.sun.star.linguistic2.XLinguServiceEventBroadcaster
- com.sun.star.lang.XInitialization
- com.sun.star.lang.XServiceDisplayName

- com.sun.star.lang.XServiceInfo
- com.sun.star.lang.XComponent

and

- com.sun.star.lang.XTypeProvider, to access your add-in interfaces from OpenOffice.org Basic, otherwise, this interface is not mandatory.

To implement a spell checker of your own, modify the sample in the following ways:

Choose a *unique* service implementation nameto distinguish your service implementation from any other. To do this, edit the string in the line

```
    public static String _aSvcImplName = "com.sun.star.linguistic2.JavaSamples.SampleSpellChecker";
```

Then, specify the list of languages supported by your service. Edit the

```
public Locale[] getLocales()
```

function and modify the

```
public boolean hasLocale( Locale aLocale )
```

function accordingly. The next step is to change the

```
private short GetSpellFailure(...)
```

as required. This function determines if a word is spelled correctly in a given language. If the word is OK return -1, otherwise return an appropriate value of the type com.sun.star.linguistic2.SpellFailure.

Check if you need to edit or remove the

```
private boolean IsUpper(...)
```

and

```
private boolean HasDigits(...)
```

functions. Consider this only if you are planning to support non-western languages and need sophisticated versions of those, or do not need them at all. Do not forget to change the code at the end of

```
public boolean isValid(...)
```

accordingly.

Supply your own version of

```
private XSpellAlternatives GetProposals(...)
```

It provides the return value for the

```
public XSpellAlternatives spell(...)
```

function call if the word was found to be incorrect. The main purpose is to provide proposals for how the word might be written correctly. Note the list ay be empty.

Next, edit the text in

```
public String getServiceDisplayName(...)
```

It should be unique but it is not necessary. If you are developing a set of services, that is, spell-checker, hyphenator and thesaurus, it should be the same for all of them. This text is displayed in dialogs to show a more meaningful text than the service implementation name.

Now, have a look in the constructor

```
public SampleSpellChecker()
```

at the property names. Remove the entries for the properties that are not relevant to your service implementation. If you make modification, also look in the file *PropChgHelper_Spell.java* in the function

```
public void propertyChange(...)
```

and change it accordingly.

Set the values of `bSCWA` and `bSWWA` to `true` only for those properties that are relevant to your implementation, thus avoiding sending unnecessary `com.sun.star.linguistic2.LinguServiceEvent` events, that is, avoid triggering spell-checking in clients if there is no requirement.

Finally, after registration of the service (see *[Chapter:Components.Deployment]*) it has to be activated to be used by the `LinguServiceManager`. After restarting OpenOffice.org, this is done in the following manner:

Open the dialog **Tools – Options – Language Settings – Writing Aids**. In the section **Writing Aids,** in the box **Available Language Modules,** a new entry with text of the Service Display Name that you chose is displayed in the implementation. Check the empty checkbox to the left of that entry. If you want to use your module, uncheck any other listed entry. If you want to make more specific settings per language, press the **Edit** button next to the modules box and use that dialog.

The Context menu of the Writer that pops up when pressing the right-mouse button over an incorrectly spelled word currently has a bug that may crash the program when the Java implementation of a spell checker is used. The spell check dialog is functioning.

Implementing a Hyphenator

A sample implementation of a hyphenator is found in the (OfficeDev/Linguistic/SampleHyphenator.java) file from the examples for linguistic.

The hyphenator implements the following interfaces:

- `com.sun.star.linguistic2.XHyphenator`
- `com.sun.star.linguistic2.XLinguServiceEventBroadcaster`
- `com.sun.star.lang.XInitialization`
- `com.sun.star.lang.XServiceDisplayName`
- `com.sun.star.lang.XServiceInfo`
- `com.sun.star.lang.XComponent`

and

- `com.sun.star.lang.XTypeProvider`, if you want to access your add-in interfaces from OpenOffice.org Basic, otherwise, this interface is not mandatory.

Aside from choosing a new service implementation name, the process of implementing the hyphenator is the same as implementing the spell checker, except that you need to implement the `com.sun.star.linguistic2.XHyphenator` interface instead of the `com.sun.star.linguistic2.XSpellChecker` interface.

You can choose a different set of languages to be supported. When editing the sample code, modify the hasLocale() and getLocales() methods to reflect the set of languages your implementation supports.

To implement the `com.sun.star.linguistic2.XHyphenator` interface, modify the functions

```
public XHyphenatedWord hyphenate(...)
public XHyphenatedWord queryAlternativeSpelling(...)
public XPossibleHyphens createPossibleHyphens(...)<
```

in the sample hyphenator source file at the stated positions.

Look in the constructor

```
public SampleHyphenator()
```

at the relevant properties and modify the

```
public void propertyChange(...)
```

function in the file (OfficeDev/Linguistic/PropChgHelper_Hyph.java) accordingly.

The rest, registration and activation is again the same as for the spell checker.

Implementing a Thesaurus

A sample implementation of a thesaurus is found in the (OfficeDev/Linguistic/SampleThesaurus.java) file from the examples for linguistic.

The thesaurus implements the following interfaces:

- `com.sun.star.linguistic2.XThesaurus`
- `com.sun.star.lang.XInitialization`
- `com.sun.star.lang.XServiceDisplayName`
- `com.sun.star.lang.XServiceInfo`
- `com.sun.star.lang.XComponent`

and

- `com.sun.star.lang.XTypeProvider`, if you want to access your add-in interfaces from OpenOffice.org Basic, otherwise, this interface is not mandatory.

For the implementation of the thesaurus, modify the sample thesaurus by following the same procedure as for the spell checker and thesaurus:

Choose a different implementation name for the service and modify the

```
public Locale[] getLocales()
```

and

```
public boolean hasLocale(...)
```

functions.

The only function to be modified at the stated position to implement the `com.sun.star.linguistic2.XThesaurus` interface is

```
public XMeaning[] queryMeanings(...)
```

Look in the constructor

```
public SampleThesaurus()
```

to see if there are properties you do not require.

Registration and activation is the same as for the spell checker and hyphenator.

6.2.4 Integrating Import and Export Filters

This section explains the implementation of OpenOffice.org import and export filter components, focussing on filter components. It is intended as a brief introduction for developers who want to implement OpenOffice.org filters for foreign file formats.

Approaches

They are several ways to get information into or out of OpenOffice.org: You can

- link against the application core
- use the document API
- use the XML file format

Each method has unique advantages and disadvantages, that are summarized briefly:

Using the core data structure and linking against the application core is the traditional way to implement filters in OpenOffice.org. The advantages of this method are efficiency and direct access to the document. However, the core implementation provides an implementation centric view of the applications. Additionally, there are a number of technical disadvantages. Every change in the core data structures or objects must be followed by corresponding changes in code that uses them. Consequently, filters need to be recompiled to match the binary layout of the application core objects. While these are manageable, albeit cumbersome, for closed source applications, this method is expected to create a maintenance nightmare if application and filters are developed separately as is customary in open source applications. Simultaneous delivery of a new application build and the corresponding filters developed by third parties looks challenging.

Using the OpenOffice.org API based on UNO is more advantageous, since it solves the technical problems indicated in the above paragraph. The idea is to read data from a file on loading and build up a document using the OpenOffice.org API, and to iterate over a document model and write the corresponding data to a file on storing. The UNO component technology insulates the filter from binary layout, and other compiler and version dependent issues. Additionally, the API is expected to be more stable than the core interfaces, and provides an abstraction from the core applications. In fact, the example filter implementation of this section makes use of this strategy and is based on the OpenOffice.org API.

The third is to import and export documents using the XML-based file format. UNO-based XML import and export components feature all of the advantages of the previous method, but additionally provide the filter implementer with a clean, structured, and fully documented view of the document. A significant difficulty in conversion between formats is the conceptual mapping from the one format to the other. From OpenOffice.org 1.1 there are XML filter components that carry out the mapping at runtime, so that filter implementers can read from XML streams when exporting and write to XML streams when importing.

The following section describes the second method using the UNO-based API. Further details on the third method, based on the generic XML format are found in the xml project of OpenOffice.org under *http://xml.openoffice.org/filter/*. The third method to create XML based filters is described afterwards.

Document API Filter Development

First, we provide an overview of the import and export process using a document API, and gain an understanding of the general concepts.

Introduction

Inside OpenOffice.org a document is represented by its document service, called *model*. On disk, the same document is represented as a file or possibly as a dynamic generated output, for example, of a database statement. We cannot assign it to a file on disk, so we call it *content* to describe it. A filter component is used to convert between these different formats.

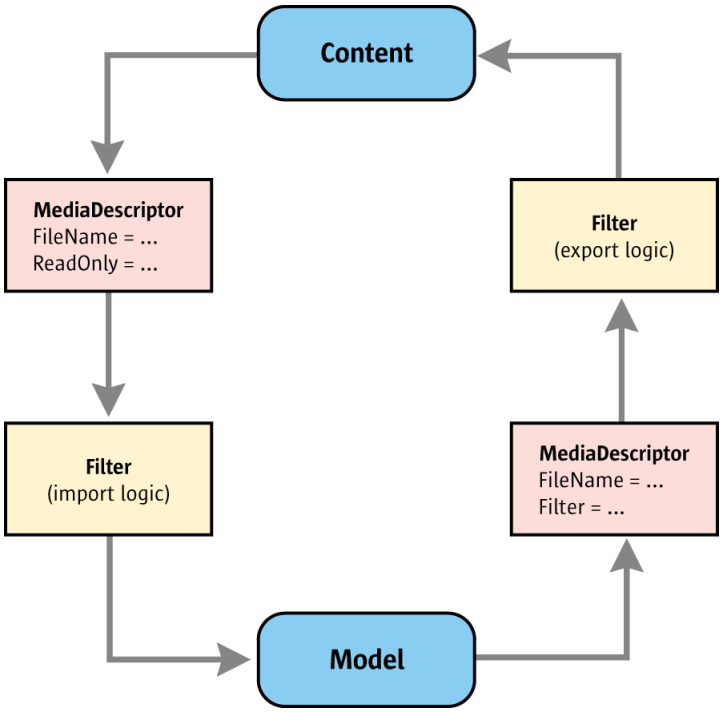


Illustration 64: Import/Export Filter Process

If you make use of UNO, this above diagram can be turned into programming reality quite easily. The three entities in the diagram, content, model, and filter, all have direct counterparts in UNO services. The services consist of several interfaces that map to a specific implementation, for example, using C++ or Java.

The filter writer creates a class that implements the `com.sun.star.document.ExportFilter` or `com.sun.star.document.ImportFilter` services, or both. To achieve this, the corresponding stream or URL is obtained from the `com.sun.star.document.MediaDescriptor`. The incoming data is then interpreted and the model is used by calling the appropriate methods. The available methods depend on the type of document as described by the document service.

For a list of available document services, refer to the section *6.1.3 Office Development - OpenOffice.org Application Environment - Using the Component Framework - Models - Document Specific Features.*

Filtering Process

Inside OpenOffice.org, the whole process of loading or saving contents is realized as a modular system that is based on UNO services. It functions generically in many components and is easily adapted to the developer's needs through the addition of custom modules or the removal of others.

Loading:

A URL or a stream is passed to `com.sun.star.frame.XComponentLoader:loadComponentFromURL()`. The load properties create a `com.sun.star.document.MediaDescriptor` that is filled with the URL or stream, and the load properties. The component loader implementation passes the information about the resource to the `TypeDetection`.

The `com.sun.star.document.TypeDetection` uses the `MediaDescriptor` to determine a unique type name that is necessary to create a filter instance at the `com.sun.star.document.FilterFactory`.

The `TypeDetection` also employs the `com.sun.star.document.ExtendedTypeDetection`that examines the given resource and confirms the unique type name determined by `TypeDetection`. The `MediaDescriptor` is updated, if necessary, and a unique type name is returned.

Finally, the component loader ensures there is a frame, or creates a new one, if necessary, and asks a frame loader service (`com.sun.star.frame.FrameLoader` or `com.sun.star.frame.SynchronousFrameLoader`) to load the resource into the frame. Its interface `com.sun.star.frame.XFrameLoader` has a method `load()` that takes a frame, the `MediaDescriptor` and an event listener, and creates a `com.sun.star.document.ImportFilter` instance at the `FilterFactory` to load the resource into the given frame. For this purpose, it calls `createInstance()` with the filter implementation name (such as com.sun.star.comp.Writer.GenericXMLFilter) or `createInstanceWithArguments()` with the implementation name and additional arguments used to initialize the filter.

Then, the loader calls `setTargetDocument()` and `filter()` on the `ImportFilter` service. The `ImportFilter` creates its results in the given target document.

Storing to a URL:

A URL or a stream is passed to `storeToURL()` or `storeAsURL()` in the interface `com.sun.star.frame.XStorable`, implemented by office documents. The store properties create a media descriptor that is filled with the URL or stream, and the store properties. The `TypeDetection` provides a unique type name that is used with the `FilterFactory` to create a `com.sun.star.document.ExportFilter`.

The `XStorable` implementation calls `setSourceDocument()` and `filter()` at the filter, which writes the results to the storage specified in the `MediaDescriptor` passed to `filter()`.



Many existing filters are legacy filters. The `XStorable` implementation does not use the `FilterFactory` to create them, but triggers filtering by internal calls.

If a URL or an already open stream takes part in the load or save process of the OpenOffice.org, the following services and operations are involved:

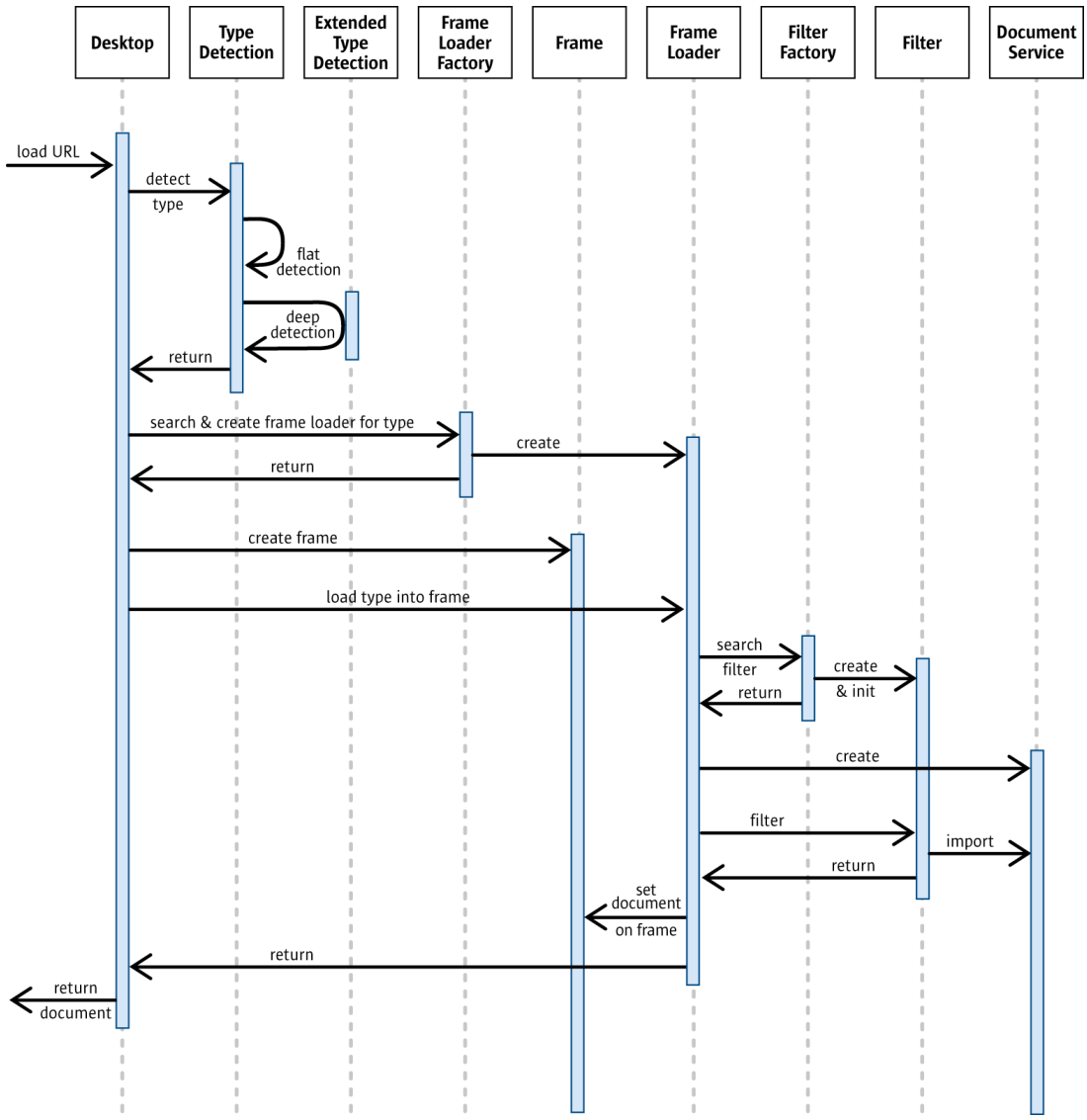


Illustration 65: General Filtering Process

In the following, the modules that participate in the loading process are discussed in detail.

MediaDescriptor

The media descriptor is an abstract description of a content specifying the *where from* and the *how* for the handling of the content to be performed. A content is also called a *medium*. Refer to *6.1.5 Office Development - OpenOffice.org Application Environment - Handling Documents - Loading Documents - MediaDescriptor* for further information. Inside the OpenOffice.org, it is realized as a sequence of `com.sun.star.beans.PropertyValue` structs as a parameter.

A descriptor is passed to various methods which are involved in the load and save process.

Every member of the process can use this descriptor and change it to update the information about the document. This descriptor is used as an [inout] parameter by `com.sun.star.document.XTypeDetection:queryTypeByDescriptor()` and `com.sun.star.document.XExtendedFilterDetection:detect()`. The `MediaDescriptor` is [in] only in `com.sun.star.frame.XComponentLoader:loadComponentFromURL()`, `com.sun.star.frame.XFrameLoader:load()` and `com.sun.star.document.XFilter:filter()`.

With methods that take the `MediaDescriptor` as [in] parameter only, a manual synchronization must be done by the outside code. The caller of a method that accepts the MediaDescriptor as [in] parameter only merges the results, for example, return values, manually into the original descriptor. The model is not available at loading time. It is the result of the load request.



It is not allowed to hold a member of this descriptor by reference longer than it is used, especially a possible stream item. For example, it would not be possible to close a stream that is still referenced by others. It is only allowed to use it directly or as a copy.



The stream part of the `MediaDescriptor` is a special item. If a stream exists, it must be used. Only if a stream does not exist, is it allowed to open a new one using the URL. The stream should be set in the `MediaDescriptor` to provide it for following users of the descriptor.
One rule exists for all: the stream inside the descriptor should be seekable. In case it is not, it makes no sense to provide it to the other members of the whole process, especially used sub-modules. On the other hand, a module can be called with a non-seekable stream from outside to perform the operation. For example, for detection or loading it should be no problem. In case a non-seekable stream comes in, but seeking is important, it must be used buffered.
Another central question is: who controls the lifetime of the stream or the stream position ? The lifetime of a non-seekable stream is controlled by the creator everytime. It has to be deleted after using. Seekable streams should be added to the MediaDescriptor and will be released by the creator of the MediaDescriptor.
Every (sub-) module must be called with a stream seeked to position 0. Of course, non-seekable streams must be newly created and unused. Internally it can do anything with this stream. Furthermore it is not necessary (or even impossible) to restore any positions. The user of the module has to do such things.

TypeDetection

Every content to be loaded must be specified, that is, the type of content represented in the OpenOffice.org must be well known in OpenOffice.org. The type is usually document type,.however, the results of active contents, for example, macros, or database contents are also described here.

A special service `com.sun.star.document.TypeDetection` is used to accomplish this. It provides an API to associate, for example, a URL or a stream with the extensions well known to OpenOffice.org, MIME types or clipboard formats. The resulting value is an internal unique type name used for further operations by using other services, for example, `com.sun.star.frame.FrameLoaderFactory`. This type name can be a part of the already mentioned `MediaDescriptor`.

It is not necessary or useful to replace this service by custom implementations.,It works in a generic method on top of a special configuration. Extending the type detection is done by changing the configuration and is described later. It is required to make these changes if new content formats are provided for OpenOffice.org, because this is the reason to integrate custom filters into the product.

ExtendedTypeDetection

Based on the registered types, flat detection is already possible, that is,. the assignment of types, for example, to a URL, on the basis of configuration data only. Tlat detection cannot always get a correct result if you imagine someone modifying the file extension of a text document from .sxw to .txt.. To ensure correct results, we need deep detection, that is, the content has to be examinedThe `com.sun.star.document.ExtendedTypeDetection` service performs this task. It is called *detector*. It gets all the information collected on a document and decides the type to assign it to. In the new modular type detection, the detector is meant as a UNO service that registers itself in the OpenOffice.org and is requested by the generic `TypeDetection` mechanism, if necessary.

To extend the list of the known content types of OpenOffice.org, we suggest implementing a detector component in addition to a filter. It improves the generic detection of OpenOffice.org and makes the results more secure.

Inside OpenOffice.org, a detector service is called with an already opened stream that is used to find out the content type. In case no stream is given, it indicates that someone else uses this service, for example, outside OpenOffice.org). It is then allowed to open your own stream by using the URL part of the `MediaDescriptor`. If the resulting stream is seekable, it should be set inside the descriptor after its position is reset to `0`. If the stream is not seekable, it is not allowed to set it. Please follow the already mentioned rules for handling streams.

FrameLoader

Frame loaders load a detected type. A visual component is expected as the result. Such visual components are:

- trivial components only implementing `com.sun.star.awt.XWindow`
- simple office components implementing the `com.sun.star.frame.Controller` service
- full featured office components implementing the `com.sun.star.document.OfficeDocument` service.

Further details are found in section *6.1.1 Office Development - OpenOffice.org Application Environment - Overview - Framework API.*

A frame loader service exist in different versions:

- `com.sun.star.frame.FrameLoader` for asynchronous
- `com.sun.star.frame.SynchronousFrameLoader` for synchronous load processes.

It can be searched or created by another service `com.sun.star.frame.FrameLoaderFactory`that is described below. The synchronous version is optional. Both services can be implemented at the same component, but the synchronous version is preferred, if it is supported.

There are two ways to extend OpenOffice.org to load a new content format:

- implementing a frame loader that uses its own internal mechanism to create the expected visual component, for example, . local file access.
- implementing a filter that does the same,but isused by a generic frame loader implementation.

Note that the first method does not work for exporting, because a loader service can not be used at save timeTo enable a content format for import and export is to provide a filter service. A generic frame loader implementation already exists in OpenOffice.orgthat uses all well known registered filters in a uniform way. So the second method is preferred.

Filter

Most of the services described before are used for loading. Normally, they are not necessary for saving, except the `MediaDescriptor`. Only filters are fixed members of both processes.

These objects also represent a service. Their task is to import or export the content of a type into or from a model. Accordingly, import filters are distinguished from export filters. It is possible to provide both functionality in the same implementation.

A filter is acquired from the factory service `com.sun.star.document.FilterFactory`. It provides a low-level access to the configuration that knows all registered filters of OpenOffice.org, supports search functionality, and creates and initializes filter components. The description of this factory and its configuration are provided below.

If a filter wants to be initialized with its own configuration data or get existing parameters of the corresponding create request, it implements the interface `com.sun.star.lang.XInitialization`. The method `initialize()` is used directly after creation by the factory and is the first request on a new filter instance. The parameter list of `initialize()` uses the following protocol:

- The first item in the list is a sequence of `com.sun.star.beans.PropertyValue` structs, that describe the configuration properties of the filter.
- All other items are directly copied from the parameter `Arguments` of the factory interface method `com.sun.star.lang.XMultiServiceFactory:createWithArguments()`.

A filter should be initialized, because one generic implementation is registered to handle different types, it must know which specialization is required. The simplest way to achieve this for the filter is to know its own configuration data, especially the unique internal name.

This information is used internally then, or it is provided by the interface `com.sun.star.container.XNamed`. An owner of a filter uses the provided name to find specific information about this component by using the `FilterFactory` service.



The interface provides functionality for reading and writing of this name. It is not allowed to change an internal filter name during runtime of OpenOffice.org, because all filter names must be unique and it is not possible for a filter instance to alter its name. Calls to `com.sun.star.container.XNamed:setName()` should be ignored or forwarded to the FilterFactory service, which knows all unique names and can solve ambigities!

This code snippet initializes a filter instance:

```
private String m_sInternalName;
public void initialize( Object[] lArguments )
         throws com.sun.star.uno.Exception
 {
         // no arguments - no initialization
         if (lArguments.length<1)
                  return;
         // Arguments[0] = own configuration data
         com.sun.star.beans.PropertyValue[] lConfig =
                  (com.sun.star.beans.PropertyValue[])lArguments[0];

         // Arguments[1..n] = optional arguments of create request
      for (int n=1; n<lArguments.length; ++n)
         {
                  ...
         }

      // analyze own configuration data for our own internal
         // filter name! Important for generic filter services,
         // which are registered more then once. They can use this
         // information to find out, which specialization of it
         // is required.
         for (int i=0; i<lConfig.length; ++i)
         {
                  if (lConfig[i].Name.equals("Name"))
                  {
                           m_sInternalName =
                                    AnyConverter.toString(lConfig[i].Value);

                           // Tip: A generic filter implementation can use this internal
                           // name at runtime, to detect which specialization of it is required.
                           if (m_sInternalName==”filter_format_1”)
                                    m_eHandle = E_FORMAT_1;
                           else
                           if (m_sInternalName==”filter_format_2”)
                                    ...
                  }
         }
 }
```

Furthermore, depending on its action a filter supports the services `com.sun.star.document.ImportFilter` for import or `com.sun.star.document.ExportFilter` for export functionality.

The common interface of both services is `com.sun.star.document.XFilter` starts or cancels the filter process. How the cancelling is implemented is an internal detail of the filter implementation, however a thread is a good solution.

On calling `com.sun.star.document.XFilter:filter()`, the already mentioned `MediaDescriptor` is passed to the service. It includes the necessary information about the content, for example, the URL or the stream, but not the source or the target model for the filter process.

Additional interfaces are part of the service description, `com.sun.star.document.XImporter` and `com.sun.star.document.XExporter` to get this information. These interfaces are used directly before the filter operation is started. A filter saves the model set by `setTargetDocument()` and `setSourceDocument()`, and uses it inside its filter operation.



The `filter()` method does not include any information about the required import or export functionality. It seems that it is not possible to implement both at the same object. The interfaces XImporter/XExporter are used to solve this conflict. Only one of them is called for one `filter()` request. So an internal flag that indicates the using of an interface helps.

This example code detects the required filter operation: (OfficeDev/FilterDevelopment/AsciiFilter/AsciiReplaceFilter.java)

```
private boolean m_bImport;

// used to tell us: "you will be used for import"
public void setTargetDocument(
 com.sun.star.lang.XComponent xDocument )
   throws com.sun.star.lang.IllegalArgumentException
{
    m_bImport = true;
}

// used to tell us: "you will be used for export"
public void setSourceDocument(
 com.sun.star.lang.XComponent xDocument )
   throws com.sun.star.lang.IllegalArgumentException
{
    m_bImport = false;
}

// detect required type of filter operation
public boolean filter(
 com.sun.star.beans.PropertyValue[] lDescriptor )
{
    boolean bState = false;
    if (m_bImport==true)
      bState = impl_import( lDescriptor );
    else
      bState = impl_export( lDescriptor );
    return bState;
}
```

The `MediaDescriptor` does not include the model, but it should include the already opened stream, true for the current implementation in OpenOffice.org. If it is there, it must be used. Only if a stream does not exist, it indicates that someone else uses this filter service, for example, outside OpenOffice.org, it creates a stream of your own by using the URL parameter of the descriptor.

In general, a filter must not change the position of an incoming stream without reading or writing data. The position inside the stream is 0. Follow the previouslymentioned rules for handling streams of the section about the MediaDescriptor above. We can make these rules easier, because currently there are no external filters used inside office. See descriptions of the chapter “MediaDescriptor” before ...)).

Filter Options

It is possible to parameterize a filter component. For example, the OpenOffice.org filter "`Text - txt - csv (StarCalc)`" needs a separator used to detect columns. This information is trans-

ported inside the `MediaDescriptor`. A special property named `FilterData` of type `any` exists. The value depends on the filter implementation and is not specified.



There is another string property named `FilterOptions`. It should be used if the flexibility of an `any` is not required. For historical reasons, a third-string property `FilterFlags` exists. It is deprecated, so it is not recommend for use.

A generic UI that uses a filter as one part of a load request does not know about special parameters. Normally, the `FilterData` are not set inside the media descriptor, therefore a filter should use default values. It should be possible to prompt the user for better values by registering another component that implements the service `com.sun.star.ui.dialogs.FilterOptionsDialog`. It is called *UIComponent*. It enables a filter developer to query for user options before the filter operation is performed. It does not show this dialog inside the filter, because any UI can be suppressed, for example, an external application uses the API of OpenOffice.org for scripting running in a hidden mode. The code that uses the filter decides if it is necessary and allowed to use the dialog. If not, the filter lives with missing parameters and uses default values. If it is not possible to have defaults, it aborts the `filter()` request returning `false`.

The UIComponent provides an interface `com.sun.star.beans.XPropertyAccess` used to set the whole `MediaDescriptor` before executing the dialog using the `FilterOptionsDialog` interface `com.sun.star.ui.dialogs.XExecutableDialog` and retrieves the changes. The user of the dialog decides if the changes are merged with the original ones or replaced. Using the whole descriptor provides the informtion about the environment in which the filter works, for example, the URL or information about preview mode. The parameters of a filter depend on it. Normally a UIComponent is shown if no FilterData or FilterOptions are part of the descriptor, so that they are added. In the case where they exist, it is necessary to change it.



If the filter programmer wants to implement a generic dialog for different filters, then he must know which of these filters the UIComponent is shown. This information exists inside the MediaDescriptor, called FilterName. The outside code which uses the dialog knows this filter alsoand should set it in the descriptor, because the implementation name of the component must be known to create the dialog. This information exists inside the configuration where it is registered for a filter.

Configuring a Filter in OpenOffice.org

As previously discussed, the whole process of loading and saving content works generically in many components and can be adapted to the needs of a user through the addition of custom modules or the removal of others. All this information about services and parameters are organized in a special configuration branch of OpenOffice.org called *org.openoffice.Office.TypeDetection*. The principal structure is shown below:

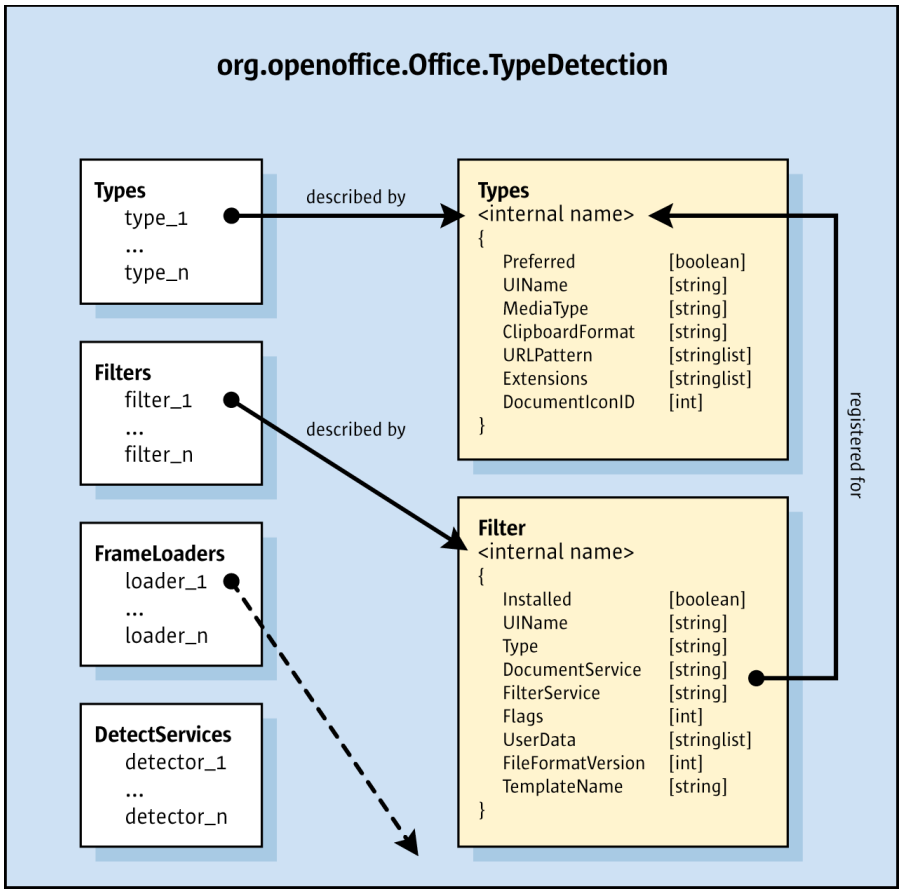


Illustration 66: Structure of org.openoffice.Office.TypeDetection Configuration Branch

As shown on the left, the file consists of lists called sets. The list items are described by the structures shown on the right to which the arrows point. It works similar to 1:n relations in a database. Every filter, frame loader, detector is registered for one or multiple types. The detection of the proper type is important for the functionality of the whole system. If the right loader or filter cannot be found, the load or save request does not produce the right results.

To extend OpenOffice.org to load or save new content formats, a new type entry is added describing the new content. Furthermore, a filter item is registered for this new type. An optional and recommended change for a detector can be done.



It is not a good idea to edit the configuration branch files directly to make these changes. It is better to use the configuration API to do so, because the format of the file may be changed in the future. The properties describing the components, such as types and filters, are always the same and are not likely to be changed or in an incompatible manner. It is better to add entries by specifying their properties using the API only.To make this easier for external programmers, this manual provides a OpenOffice.org Basic script that is used for that purpose called *regfilter.bas*.

The work to be done by the filter programmer is to provide an ini file that includes the properties and start the basic script inside OpenOffice.org. The script reads the file and uses it to change the configuration package. These changes are done for the user layer of the configuration, so it is possible to restore the original state. There is also an example ini file in the samples folder for this manual that can be used for your own purposes called *regfilter.ini*.

General Notes

In OpenOffice.org, there are services providing a special API to access the underlying configuration repository. Most of these services support container functionality and allow read access whereas some services offer write access also . During runtime, every configuration item, such as type, filter, and detector, is represented as a sequence of `com.sun.star.beans.PropertyValue` structs. The next sections describe the names and values of those structures.

Necessary Steps

To extend OpenOffice.org by new content formats, use the following steps:

1. Implement a filter component. It must be able to load or save the type it is registered for. For access to the office, only the API of the document service or universal content provider keeps the filter compatible with new versions of OpenOffice.org.
2. Provide an implementation of a `com.sun.star.document.ExtendedTypeDetection` service to analyze a given content. It must return an internal type name representing the type or an empty value for unknown formats.
3. Add a filter options dialog if the implemented filter requires additional parameters. Keep it separate from the filter and change the given `MediaDescriptor` based on user input. Document the parameters so that an external script programmer can use this information to provide proper values to the `MediaDescriptor`.
4. Register the component libraries as UNO services inside OpenOffice.org. This is done by the mechanism described in the chapter *4.9 Writing UNO Components - Deployment Options for Components*.
5. Adapt the configuration branch *org.openoffice.Office.TypeDetection* so that it knows these new components. Use OpenOffice.org Basic script *regfilter.bas* that is provided as an additional tool in this chapter. It requires an ini file that is specified inside the subroutine `Main` of the script and has to be adjusted for your own purposes. It is well documented, and uses the names and value types described in this manual.

Properties of a Type

Every type inside OpenOffice.org is specified by the properties shown in the table below. These values are accessible at the previously mentioned service `com.sun.star.document.TypeDetection` using the interface `com.sun.star.container.XNameAccess`. Write access is not available here. All types are addressed by their internal names.

Properties of a Document Type, available at TypeDetection	
`Name`	string. The internal name of a type must be unique and is also used as a list entry. It contains any special characters, but they must been coded.
`UIName`	string. Displays the type at the user interface under a localized name. You must assign a value for a language, thus supporting CJK versions. All Unicode characters are permitted here.
`MediaType`	string. Describes the MIME type of the contents. The reason is that the internal names can be altered at any time without affecting the process.
`ClipboardFormat`	string. The format is a unique description of this type for use in clipboards.

Properties of a Document Type, available at TypeDetection	
URLPattern	sequence<string>. Important components of a type are the patterns. They enable the support of your own URL schemata, for example, in OpenOffice.org "private:factory/swriter" for opening an empty text document. The wildcards '*' or '?' are supported here.
Extensions	sequence<string>. The type of a content can be derived from its URL by its extension. In most cases, the flat detection depends on them alone.
Preferred	boolean. Since file extensions cannot always be assigned to a unique type, this flag was introduced. It indicates the preferred type for a group of types with similar properties, otherwise, the first match is used.
DocumentIconID	int. You can assign an icon to a type. To do this, the ID is used as reference to a resource. This feature is currently not supported in OpenOffice.org.

Properties of an ExtendedTypeDetection Service

In contrast to filters or frame loaders, the ExtendedTypeDetection has no configuration API on top of its configuration data. The normal configuration API of OpenOffice.org has to be used, as described in *[Chapter:Config]*. The configuration set `org.openoffice.Office.TypeDetection/DetectServices` could be used, but it is better to use the already mentioned basic macro *regfilter.bas* in combination with *regfilter.ini*. Such detector services are used automatically during type detection of content. A detector service is addressed by its UNO implementation name.

Property Name	Description
ServiceName	string. This must be a valid UNO implementation name. This field cannot contain the service name, because this value must be unique, otherwise it would be impossible to distinguish more than one registered entry, for a service name is not unique. This value is also an entry in the corresponding configuration list.
Types	sequence<string>. A list of type names recognized by this service that makes it possible to write a servicethat detects more than one type.

Properties of a Filter

Every filter is registered for only one type . Multiple registrations are to be done by multiple configuration entries. One type is handled by more than one filter. Flags also regulate the use of the preferred filter. A filter is described by the following properties:

Property Name	Description
Name	string. The internal name of a filter must be unique and is also used as list entry. It contains special characters, but they must be encoded.
UIName	string. A filter should be able to show a localized name in selection dialogs. You must assign a value for a language, thus supporting CJK versions. All Unicode characters are permitted here.
Installed	boolean. This flag indicates the installation status of a filter. A filter is generally registered equally for all users. In a network installation you should deactivate this for certain groups or single users. Note: A filter works only if the component library has already been registered in OpenOffice.org.

Property Name	Description
Order	int. This number shows filters in a user defined order. Valid values are greater then 0. If the number is set to 0, sorting is done alphabetically by the UIName property of the filter. The same applies to filters that have the same Order value.
Type	string. A filter must register itself for the type it can handle. Multiple assignments are not allowed. Multiple configuration entries must be created, one for every supported type.
DocumentService	string. Describes the component for which the filter operates,For example, "com.sun.star.text.TextDocument", depending upon the use., This is considered the output or goal of the filter process. A UNO service name is expected. Note: The implementation name cannot be used here, the generic type of the document is needed.
FilterService	string. This is the UNO implementation name of the filter. It should be clear that this field can not contain the service name of a filter, otherwise OpenOffice.org could not distinguish more then one registered filter.
UIComponent	string. Describes an implementation of a UI dialog used by the filter to let the user modify certain properties for filtering. For example, the "Text - txt - csv (StarCalc)" needs information about the used column separators to load data. To distinguish between different implementations, it must be the real UNO implementation name, not a service name.
Flags	int. Describes the filter, as shown in the table below. This is where, the organization into import and export filters takes place. Note that external filters must set the ThirdParty flag to be detected.
UserData	sequence<string>. Some filters need to store more configuration data than usual. This is realized through this entry. The format of the string list is not restricted.
FileFormatVersion	int. Indicates a version number of a document that can be edited by this filter.
TemplateName	string. The name of a template file for importing styles. It is a special feature for importing documents only and not useable for export. Every OpenOffice.org document service knows default styles. If this TemplateName is set, it merges these default styles with the styles of the template, and the template styles are merged with all styles of a document that is imported by this filter.

Most functionality of a Filter is listed by its flags. They are necessary to prevent a filter from being displayed in a UI, and to classify import and export, or internal a nd external filters, and prefer some filters to others. Currently supported flags are:

Name	Value	Description
Import	0x00000001 h	This filter supports the specification of a com.sun.star.document.ImportFilter and is used for loading content.
Export	0x00000002 h	This filter supports the specification of a com.sun.star.document.ExportFilter and is used for saving content.
Template	0x00000004 h	These filters are specialized to handle template formats. By default, a filtered document is used as a template to create a new document .
Internal	0x00000008 h	This filter should never be shown on any UI and not be available.
OwnTemplate	0x00000010 h	Templates used with the template API of OpenOffice.org and it supports the internal template features. For older versions, it is useable for internal content formats only.

Name	Value	Description
Own	0x00000020 h	Tag the intrinsic content formats of OpenOffice.org based on OLE storage or zip packages.
Alien	0x00000040 h	A filter with this flag is not fully compatible with the current document format. It is unclear what document features will be lost during saving. This flag decides if a warning box on saving has to be shown.
UsesOptions (deprecated)	0x00000080 h	This filter could be customized during processing. Older versions of OpenOffice.org used it to customize the "SaveAs" dialog. Newer versions uses the filter property "UIComponent" to tell if a filter provides filter options.
Default	0x00000100 h	Mark a filter as the default filter for saving. Only one filter in an application module, distinguished through the Document-Service property, has this flag set.
NotInFileDialog	0x00001000 h	Suppress display of a filter in file open and save dialogs.
NotInChooser	0x00002000 h	Suppress display of a filter in UI elements for choosing filters.
ThirdParty	0x00080000 h	These filters are developed by external parties. For historical reasons, the filter detection of OpenOffice.org differentiates between old internal and new external ones, because the former are not UNO based and are used differently.
Preferred	0x10000000 h	If more than one filter is registered for the same type, this flag prefers one of them at loading time if the user does not select a specific filter. In contrast to the Default flag, it does not depend on the application module, but there can only be one preferred filter for a type.



Besides these filter flags there are other flags existing that are used currently, but are not documented here. Use documented flags only.

The service com.sun.star.document.FilterFactory provides these data. It supports read access by using the interface com.sun.star.container.XNameAccess. All items are addressed by their internal names. The return value is represented as a list of type com.sun.star.beans.PropertyValue structures. It uses the filter properties shown above.

Another aspect of this service is the factory interface com.sun.star.lang.XMultiServiceFactory. It creates filter instances using an internal type, or an internal filter name directly. Using a type name searches for a suitable filter and creates, initializes and returns it. Using a filter name directly follows the algorithm shown in the box below. Note that creation of filters is possible for external ones only that have set the FilterService property. Most of the current filters of OpenOffice.org are internal filters, implemented as local code, but not as a UNO service. They can not be created by this FilterFactory. It is possible to ask only for their properties.



Direct creation of a filter instance is only possible using a special argument in the createInstanceWithArguments() call of the interface XMultiServiceFactory. To do so, a com.sun.star.beans.PropertyValue FilterName with the internal name of the requested filter as value must be used. Otherwise, the service specifier, that is, the first argument of the create call, is interpreted as an internal type name. It will be used to search a suitable, preferred filter that will be created. It is a combination of searching and creation. Future implementations will split that to make it clearer. In future implementations, a registered filter must be searched through the provided query mechanism and created by using this factory interface.

Properties of a FrameLoader

OpenOffice.org distinguishes asynchronous (`com.sun.star.frame.FrameLoader`) and synchronous (`com.sun.star.frame.SynchronousFrameLoader`) frame loader implementations, but the configuration does not recognize that. The interface is supported by the loader is detected at runtime , the synchronous interface being preferred. The following properties describe a loader:

Properties of a FrameLoader	
`Name`	string. This must be a valid UNO implementation name. It should be obvious that this field can not contain the service name, because this value must be unique. Otherwise OpenOffice.org could not distinguish more than one registered entry, for there can be several implementations for a service name. This value is also an entry in the corresponding configuration list.
`UIName`	string. Displays the loader at a localized user interface. You must assign a value for a language, thus supporting CJK versions. All Unicode characters are permitted.
`Types`	sequence<string>. A list of type names recognized by this service You can also implement and register loader for groups of types.

The service `com.sun.star.frame.FrameLoaderFactory` makes this data available. It uses the same mechanism as the `com.sun.star.document.FilterFactory`, that is, an interface for data access, `com.sun.star.container.XNameAccess`, and another one for creation of such a `FrameLoader`, `com.sun.star.lang.XMultiServiceFactory`.

There are other properties than the properties described, for example, for the ContentHandler. They are not necessary for the environment of filters, or loading and saving documents, so they are not described. Additional information is found at *http://framework.openoffice.org*.

There is one entry in the configuration, used as a fallback if a registered item is not found, the generic `FrameLoader`. It is not necessary for an external developer to provide a frame loader to add support for an unknown document format to OpenOffice.org. It is enough to register a new filter component that is used by this special loader in a generic manner.

XML Based Filter Development

Introduction

This chapter outlines the development of XML based filtering components that use the XML filter adaptor framework. The XML filter adaptor is a generic `com.sun.star.document.XFilter` implementation. It has been designed to be reusable, and to supply a standard method of designing and referencing XML based import and export filters. The XML filter adaptor does not perform any of the filtering functionality itself, but instead is used to instantiate a filtering component.

The advantage of the XML filter adaptor framework is that you do not have to work with document models to create a document from an import file, nor do you have to iterate over a document model to export it to a different file format. Rather, you can use the OpenOffice.org XML file format to import and export. When importing, you parse your import file and send OpenOffice.org XML to the filter adaptor, which creates a document for you in the GUI. When exporting, the office sends a description of the current document as OpenOffice.org XML, so that you can export without having to iterate over a document model.

The course of action during export and import can be described as follows: when a user clicks **File-Open**, or some UNO code calls `loadComponentFromURL()`, the office looks in the type detection configuration to identify an import filter and optionally checks the file format by doing some deep detection. It instantiates the import filter it finds and uses its method `importer()` to pass a

`MediaDescriptor` for the source, a specialized XML document handler for OpenOffice.org XML, and user data. The import filter has to read the import source and deliver OpenOffice.org XML to the document handler received in the call to `importer()`, emulating a SAX parser that calls the parser callback functions.

Similarly, the office instantiates an export filter after clicking **File-Save (As)** or a call to `storeXXX()`, and uses its method `exporter()` to pass a target location and user data. In this case, the office expects the export filter to be a `com.sun.star.xml.sax.XDocumentHandler`, which is able to handle OpenOffice.org XML. The office creates an export stream with OpenOffice.org XML, and parses this XML so that the export filter receives the SAX callbacks and can translate them to whatever is necessary, writing the result to the target received in the call to `com.sun.star.xml.XExportFilter:exporter()`.

Components of a Filter

For a filter of this type to operate, three things are necessary.

1. The XML filter adaptor.
2. A filtering component that implements the required interfaces.
3. A valid filter and type definition.

Both the XML filter adaptor and the filtering component are UNO components that can be instantiated through the `com.sun.star.lang.XMultiServiceFactory:createInstance()` method. Since the XML filter adaptor is generic, the filtering component is all that needs to be implemented. Once this has been done, the *TypeDetection.xcu* file can be expanded to include the newly created filter definition.

Writing the Filtering Component

The filtering component must implement the following interfaces as described by the `com.sun.star.xml.ImportFilter` service and the `com.sun.star.xml.ExportFilter` service:

Importer:
`com.sun.star.xml.XMLImportFilter`

Exporter:
`com.sun.star.xml.XMLExportFilter` and `com.sun.star.xml.sax.XDocumentHandler`

XImportFilter

The service`com.sun.star.xml.XMLImportFilter` defines an interface with the following method:

```
boolean importer(
 [in] sequence< com::sun::star::beans::PropertyValue > aSourceData,
 [in] com::sun::star::xml::sax::XDocumentHandler xDocHandler,
 [in] sequence< string > msUserData )
```

`aSourceData` is a `MediaDescriptor`, which can be used to obtain the following information:

- An `XInputStream`
  This is a stream that is attached to the source to be read. This can be a file, or some other data source.
- `Filename`
  This is the name of the file on the disk, that the input stream comes from.
- `Url`
  This is a url describing the location being read.

`xDocHandler` is a `SAX` event handler that can be used when parsing an `XInputStream`, which may or may not contain OpenOffice.org XML. Before this stream can be read by OpenOffice.org, it will need to be transformed into OpenOffice.org XML.

`msUserData` is an array of `Strings` , that contains the information supplied in the `UserData` section of the `Filter` definition in the *TypeDetection.xcu* file.

XExportFilter

The `com.sun.star.xml.XExportFilter` defines an interface with the following method:

```
boolean exporter(
[in] sequence< com::sun::star::beans::PropertyValue > aSourceData,
[in] sequence< string > msUserData )
```

`aSourceData` and `msUserData` contain the same type of information as in the importer, except that the `MediaDescriptor` contains an `XOutputStream` , which can be used to write to.

XDocumentHandler

When the export takes place, the new Filtering component must also be an `XDocumentHandler` , to allow the output based on SAX events to be filtered, if required. For this reason, an `XDocumentHandler` is not passed to the exporter, and any exporter that is used by the XML filter adaptor must implement the `com.sun.star.xml.sax.XDocumentHandler` interface.

The Importer

Evaluating XImportFilter Parameters

The writing of an importer usually starts with extracting the required variables from the `Mediadescriptor` and the `userData` . These variables are required for the filtering component to operate correctly. Depending on the requirements of the individual filter, the first thing to do is to extract the information from the `MediaDescriptor`, referred to as `aSourceData` in the interface definition. This can be achieved as follows:

Get the number of elements in the `MediaDescriptor`

```
sal_Int32 nLength = aSourceData.getLength();
```

Iterate through the `MediaDescriptor` to find the information needed: an input stream, a file name, or a URL.

```
for (sal_Int32 i = 0; i < nLength; i++) {
         if (pValue[i].Name.equalsAsciiL (RTL_CONSTASCII_STRINGPARAM("InputStream")))
                  pValue[i].Value >>= xInputStream;
         else if (pValue[i].Name.equalsAsciiL(RTL_CONSTASCII_STRINGPARAM("FileName")
                  pValue[i].Value >>= sFileName;
         else if (pValue[i].Name.equalsAsciiL(RTL_CONSTASCII_STRINGPARAM("URL")))
                  pValue[i].Value >>= sURL;
}
```

The `msUserData` parameter passed to `importer()` contains information that defines how the filter operates, so this information must be referenced as required.

Importer Filtering

An `XInputStream` implementation has now been obtained that contains all of the information you want to process. From the filtering perspective, you can just read from this stream and carry out whatever processing is required in order for the input to be transformed into OpenOffice.org XML. Once this has been done, however, you need to write the result to where it can be parsed into OpenOffice.org's internal format. A `Pipe` can be used to achieve this. A `Pipe` is a form of buffer that can be written to and read from. For the `importer`, read from the `XInputStream` that

was extracted from the MediaDescriptor, and once the filtering has taken place, write to a Pipe that has been created. This Pipe can be read from when it comes to parsing. This is how the Pipe is created:

```
Reference <XInterface> xPipe;

// We Create our pipe
xPipe= XflatXml::xMSF->createInstance(OUString::createFromAscii("com.sun.star.io.Pipe"));

// We get an inputStream to our Pipe
Reference< com::sun::star::io::XInputStream > xPipeInput (xPipe,UNO_QUERY);

// We get an OutputStream to our Pipe
Reference< com::sun::star::io::XOutputStream > xTmpOutputStream (xPipe,UNO_QUERY);
```

The XInputStream can be read from, and the XOutputstream can be written to.

Parsing the Result

Once the desired OpenOffice.org XML has been produced and written to the XOutputStream of the Pipe, the XinputStream of the Pipe can be parsed with the aid of the XdocumentHandler.

```
// Create a Parser
const OUString sSaxParser(RTL_CONSTASCII_USTRINGPARAM("com.sun.star.xml.sax.Parser"));
Reference < com::sun::star::xml::sax::XParser > xSaxParser(xMSF->createInstance(sSaxParser), UNO_QUERY);

// Create an InputSource using the Pipe
com::sun::star::xml::sax::InputSource aInput;
aInput.sSystemId = sFileName;             // File Name
aInput.aInputStream = xPipeInput;         // Pipe InputStream

// Set the SAX Event Handler
xSaxParser->setDocumentHandler(xHandler);

// Parse the result
try {
  xSaxParser->parseStream(aInput);
}
catch( Exception &exc){
  // Perform exception handling
}
```

Assuming that the XML was valid, no exceptions will be thrown and the importer will return true. At this stage, the filtering is complete and the imported document will be displayed.

The Exporter

Evaluating XExportFilter Parameters

The exporter() method operates in much the same way as importer(), except that instead of the exporter using a provided XDocumentHandler, it is itself a com.sun.star.xml.sax.XDocumentHandler implementation.

When the exporter() method is invoked, the necessary variables need to be extracted for use by the filter. This is the same thing that happens with the importer, except that the MediaDescriptor contains an XOutputStream, instead of the importer's XInputStream. Once the variables have been extracted (and—in some cases—a Pipe has been created) the exporter() method returns. It does not carry out the filtering at this stage.



The pipe is only necessary if the output needs to be processed further after being processed by the XDocumentHandler. Otherwise, the result from the XDocumentHandler implementation can be written directly to the XOutputStream provided. For instance, this is the case with a FlatXML filter.

Exporter Filtering

After the exporter() method returns, the XML filter adaptor then invokes the com.sun.star.xml.sax.XDocumentHandler methods to parse the XML output.

For the filtering, the `com.sun.star.xml.sax.XDocumentHandler` implementation is used. This consists of a set of SAX event handling methods, which define how particular XML tags are handled. These methods are:

```
startDocument(){
}
endDocument(){
}
startElement(){
}
endElement(){
}
charactors(){
}
ignorableWhitespace(){
}
processingInstruction(){
}
setDocumentLocator(){
}
```

The result of this event handling can be processed and written to the `XOutputStream` that was extracted from the `MediaDescriptor`.

Configuration

For OpenOffice.org to be able to make use of this filtering component, the filter and the type that it handles must be defined in the *TypeDetection.xcu* file.

The type section defines certain file types and their extensions. The filter section contains the actual filter definition.

Below is an example of a type and filter definition in the *TypeDetection.xcu* file. It describes a `PocketWord` filter.

```
<!-- Type section -->
<node oor:name="writer_PocketWord_File" oor:op="replace">
    <prop oor:name="UIName">
        <value xml:lang="en-US">Pocket Word</value>
    </prop>
    <prop oor:name="Data">
        <value>0,,,,psw,20002,</value>
    </prop>
</node>

<!-- Filter section -->
<node oor:name="PocketWord File" oor:op="replace">
    <prop oor:name="UIName">
        <value xml:lang="en-US">Pocket Word</value>
   </prop>
   <prop oor:name="Data">
        <value>
                0,
                writer_PocketWord_File,
                com.sun.star.text.TextDocument,
                com.sun.star.comp.Writer.XmlFilterAdaptor,
                524355,
                com.sun.star.documentconversion.XMergeBridge;
                classes/pocketword.jar;
                com.sun.star.comp.Writer.XMLImporter;
                com.sun.star.comp.Writer.XMLExporter;
                staroffice/sxw;application/x-pocket-word,
                0,
                ,
        </value>
   </prop>
   <prop oor:name="Installed" oor:type="xs:boolean">
        <value>true</value>
   </prop>
</node>
```

The type section defines a type `writer_PocketWord_File` with `"Pocket Word"` as `UIName` in **File - Open**. The file extension of this type is specified as *.psw* in the `Data` property.

The filter section defines the same UI name `"Pocket Word"` for the filter, and a number of settings in the `Data` property, simply separated by commas:

Type
This is the id of the file type definition that defines the type of file that this filter can handle, in this case "writer_PocketWord_File". This value refers to the oor:name of the type section.

Office application
This is the application that will be used to open the document, in this case Writer ("com.sun.star.text.TextDocument").

Filter Component
This is the component that OpenOffice.org will initialize when importing or exporting, in this case "com.sun.star.comp.Writer.XmlFilterAdaptor".

User Data
Section containing the filtering component that the XML filter adaptor will initialize and use for filtering. In this example
"com.sun.star.documentconversion.XMergeBridge;classes/pocketword.jar;com.sun.star.comp.Writer.XMLImporter;
com.sun.star.comp.Writer.XMLExporter;staroffice/sxw;application/x-pocket-word"

From this example, you see that this Filter uses the XML filter adaptor. When the XML filter adaptor initializes, it initializes the XMergeBridge that is specified in the UserData section. The rest of the information in the UserData section has been included for use by the XMergeBridge filtering component, including the com.sun.star.comp.Writer.XMLImporter, which will be used by the XML filter adaptor to create the XDocumentHandler when importing.

Sample Filter Component Implementations

There are currently three filtering components which use the XML filter adaptor.

The first one is the XMergeBridge. This has been created as a means of linking the XMerge Small Device filter framework with OpenOffice.org. This means that any available XMerge plugin, can also be used as a OpenOffice.org filter. This is currently hosted within the XMerge project in openoffice cvs at

xml/xmerge/java/org/openoffice/xmerge/xmergebridge

The final two are a Java and a C++ implementation of a *Flat OpenOffice.org XML* reader and writer. These are intended to be sample filter component implementations, and offer a skeleton filter component that can be expanded upon by developers wishing to create their own filtering components. These are temporarily hosted in cvs at

xml/xmerge/java/org/openoffice/xmerge/xmergebridge/FlatXml

Additional Components

In order for Java based components to operate effectively, a set of wrapper classes have been added to the *javaunohelper* package. These files allow for an XInputStream or an XOutputStream to be accessed using the same methods as a normal Java InputStream or OutputStream. These classes are located in the *javaunohelper* package at

```
com.sun.star.lib.uno.adapter.XInputStreamToInputStreamAdapter
com.sun.star.lib.uno.adapter.XInputStreamToInputStreamAdapter
```

For more information on the use of these helper classes, see the flatxmljava example.

XML Filter Detection

The number of XML files that conform to differing DTD specifications means that a single filter and file type definition is insufficient to handle all of the possible formats available. In order to

allow OpenOffice.org to handle multiple filter definitions and implementations, it is necessary to implement an additional filter detection module that is capable of determining the type of XML file being read, based on its `DocType` declaration.

To accomplish this, a filter detection service `com.sun.star.document.ExtendedTypeDetection` can be implemented, which is capable of handling and distinguishing between many different XML based file formats. This type of service supersedes the basic *flat* detection, which uses the file's suffix to determine the Type, and instead, carries out a *deep* detection which uses the file's internal structure and content to detect its true type.

Requirements for Deep Detection

There are three requirements for implementing a deep detection module that is capable of identifying one or more unique XML types. These include:

- An extended type definition for describing the format in more detail (*TypeDetection.xcu*).
- A `DetectService` implementation.
- A `DetectService` definition (*TypeDetection.xcu*).

Extending the File Type Definition

Since many different XML files can conform to different DTDs, the type definition of a particular XML file needs to be extended. To do this, some or all of the `DocType` information can be contained as part of the file type definition. This information is held as part of the `Clipboard-Format` property of the type node. A unique namespace or preface identifies the String at this point in the sequence as being a `DocType` declaration.

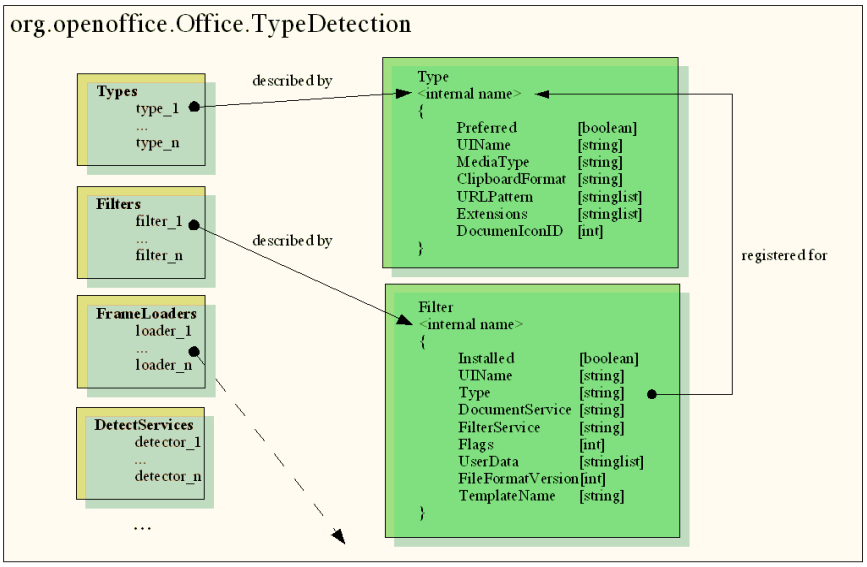


Illustration 67

Sample Type definition:

```
<node oor:name="writer_DocBook_File" oor:op="replace">
    <prop oor:name="UIName">
        <value XML:lang="en-US">DocBook</value>
    </prop>
    <prop oor:name="Data">
        <value> 0,
                ,
                doctype:-//OASIS//DTD DocBook XML V4.1.2//EN,
                ,
```

```
                XML,
                20002,
        </value>
    </prop>
</node>
```

The ExtendedTypeDetection Service Implementation

In order for the type detection code to function as an `ExtendedTypeDetection` service, you must implement the `detect()` method as defined by the `com.sun.star.document.XExtendedFilterDetection` interface definition:

```
string detect( [inout]sequence<com::sun::star::beans::PropertyValue > Descriptor );
```

This method supplies you with a sequence of `ProptertyValue`s from which you can use to extract the current `TypeName` and the `URL` of the file being loaded:

```
::rtl::OUString SAL_CALL FilterDetect::detect(com::sun::star::uno::Sequence<
com::sun::star::beans::PropertyValue >& aArguments ) throw (com::sun::star::uno::RuntimeException) {
const PropertyValue * pValue = aArguments.getConstArray();
sal_Int32 nLength;
::rtl::OString resultString;
nLength = aArguments.getLength();
for (sal_Int32 i = 0; i < nLength; i++) {
        if (pValue[i].Name.equalsAsciiL(RTL_CONSTASCII_STRINGPARAM("TypeName"))) {
        }
        else if (pValue[i].Name.equalsAsciiL(RTL_CONSTASCII_STRINGPARAM("URL"))) {
                pValue[i].Value >>= sUrl;
        }
}
```

Once you have the `URL` of the file, you can then use it to create a `::ucb::Content` from which you can open an `XInputStream` to the file:

```
Reference< com::sun::star::ucb::XCommandEnvironment > xEnv;
::ucb::Content aContent(sUrl,xEnv);
xInStream = aContent.openStream();
```

You can now use this `XInputStream` to read the header of the file being loaded. Because the exact location of the `DocType` information within the file is not known, the first 1000 bytes of information will be read:

```
::rtl::OString resultString;
com::sun::star::uno::Sequence< sal_Int8 > aData;
long bytestRead =xInStream->readBytes (aData,  1000);
resultString=::rtl::OString(
(const sal_Char *)aData.getConstArray(),bytestRead) ;
```

Once you have this information, you can start looking for a type that describes the file being loaded. In order to do this, you need to get a list of the types currently supported:

```
Reference <XNameAccess> xTypeCont(mxMSF->createInstance(OUString::createFromAscii(
                                  "com.sun.star.document.TypeDetection" )),UNO_QUERY);
Sequence <::rtl::OUString> myTypes= xTypeCont->getElementNames();
nLength = myTypes.getLength();
```

For each of these types, you must first determine whether the `ClipboardFormat` property contains a `DocType`:

```
Loc_of_ClipboardFormat=...;
Sequence<::rtl::OUString> ClipboardFormatSeq;
Type_Props[Loc_of_ClipboardFormat].Value >>=ClipboardFormatSeq ;
while() {
        if(ClipboardFormatSeq.match(OUString::createFromAscii("doctype:") {
                //if it contains a DocType, start to compare to header
        }
}
```

All of the possible `DocType` declarations of the file types can be checked to determine a match. If a match is found, the type corresponding to the match is returned. If no match is found, an empty string is returned. This will force OpenOffice.org into flat detection mode.

TypeDetection.xcu DetectServices Entry

Now that you have created the `ExtendedTypeDetection` service implementation, you need to tell OpenOffice.org when to use this service.

First create a `DetectServices` node, unless one already exists, and then add the information specific to the detection service that has been implemented, that is, the name of the service and the file types that use it.

```
<node oor:name="DetectServices">
<node oor:name="com.sun.star.comp.filters.XMLDetect" oor:op="replace">
        <prop oor:name="ServiceName">
                <value XML:lang="en-US">com.sun.star.comp.filters.XMLDetect</value>
        </prop>
        <prop oor:name="Types">
                <value>writer_DocBook_File</value>
                <value>writer_Flat_XML_File</value>
        </prop>
</node>
</node>
```

6.2.5 Number Formats

Number formats are template strings consisting of format codes defining how numbers or text appear, for example,, whether or not to display trailing zeroes, group by thousands, separators, colors, and how many decimals are displayed. This does not include any font attributes, except for colors. They are found wherever number formats are applied, for example, on the **Numbers** tab of the **Format – Cells** dialog in spreadsheets.

Number formats are defined on the document level. A document displaying formatted values has a collection of number formats, each with a unique index key within that document. Identical formats are not necessarily represented by the same index key in different documents.

Managing Number Formats

Documents provide their formats through the interface `com.sun.star.util.XNumberFormatsSupplier`that has one method `getNumberFormats()` that returns `com.sun.star.util.NumberFormats`. Using `NumberFormats`, developers can read and modify number formats in documents, and also add new formats.

You have to retrieve the `NumberFormatsSupplier` as a property at a few objects from their `com.sun.star.beans.XPropertySet` interface, for example, from data sources supporting the `com.sun.star.sdb.DataSource` service and from database connections supporting the service `com.sun.star.sdb.DatabaseEnvironment`, or `com.sun.star.sdb.DatabaseAccess`. In addition, all UNO controls offering the service `com.sun.star.awt.UnoControlFormattedFieldModel` have a `NumberFormatsSupplier` property.

NumberFormats Service

The `com.sun.star.util.NumberFormats` service specifies a container of number formats and implements the interfaces `com.sun.star.util.XNumberFormatTypes` and `com.sun.star.util.XNumberFormats`.

XNumberFormats

`NumberFormats` supports the interface `com.sun.star.util.XNumberFormats`. This interface provides access to the number formats of a container. It is used to query the properties of a

number format by an index key, retrieve a list of available number format keys of a given type for a given locale, query the key for a user-defined format string, or add new format codes into the list or to remove formats.

```
com::sun::star::beans::XPropertySet getByKey ( [in] long nKey )

sequence< long > queryKeys ( [in] short nType,
                [in] com::sun::star::lang::Locale nLocale,
                [in] boolean bCreate )
long queryKey ( [in] string aFormat,
                [in] com::sun::star::lang::Locale nLocale,
                [in] boolean bScan )

long addNew ( [in] string aFormat, [in] com::sun::star::lang::Locale nLocale )
long addNewConverted ( [in] string aFormat, [in] com::sun::star::lang::Locale nLocale,
              [in] com::sun::star::lang::Locale nNewLocale )

void removeByKey ( [in] long nKey )

string generateFormat ( [in] long nBaseKey, [in] com::sun::star::lang::Locale nLocale,
              [in] boolean bThousands, [in] boolean bRed, [in] short nDecimals, [in] short nLeading )
```

The important methods are probably `queryKey()` and `addNew()`. The method `queryKey()` finds the key for a given format string and locale, whereas `addNew()` creates a new format in the container and returns its key for immediate use. The `bScan` is reserved for future use and should be set to `false`.

The properties of a single number format are obtained by a call to `getByKey()` which returns a `[IDL:com.sun.star.util.NumberFormatProperties]` service for the given index key.

XNumberFormatTypes

The interface `com.sun.star.util.XNumberFormatTypes` offers functions to retreive the index keys of specific predefined number format types. The predefined types are addressed by constants from `com.sun.star.util.NumberFormat`.The `NumberFormat` contains values for predefined format types, such as PERCENT, TIME, CURRENCY, and TEXT.

```
long getStandardIndex ( [in] com::sun::star::lang::Locale nLocale )
long getStandardFormat ( [in] short nType,
                [in] com::sun::star::lang::Locale nLocale )
long getFormatIndex ( [in] short nIndex,
                [in] com::sun::star::lang::Locale nLocale )

boolean isTypeCompatible ( [in] short nOldType, [in] short nNewType )

long getFormatForLocale ( [in] long nKey,
                [in] com::sun::star::lang::Locale nLocale )
```

In most cases you will need `getStandardFormat()`. It expects a type constant from the NumberFormat group and the locale t to use, and returns the key of the corresponding predefined format.

Applying Number Formats

To format numeric values, an `XNumberFormatsSupplier` is attached to an instance of a `com.sun.star.util.NumberFormatter`, available at the global service manager. For this purpose, its main interface `com.sun.star.util.XNumberFormatter` has a method `attachNumberFormatsSupplier()`. When the `XNumberFormatsSupplier` is attached, strings and numeric values are formatted using the methods of the `NumberFormatter`. To specify the format to apply, you have to get the unique index key for one of the formats defined in `NumberFormats`. These keys are available at the `XNumberFormats` and `XNumberFormatTypes` interface of `NumberFormats`.

Numbers in documents, such as in table cells, formulas, and text fields, are formatted by applying the format key to the NumberFormat property of the appropriate element.

NumberFormatter Service

The service `com.sun.star.util.NumberFormatter` implements the interfaces `com.sun.star.util.XNumberFormatter` and `com.sun.star.util.XNumberFormatPreviewer`.

XNumberformatter

The interface `com.sun.star.util.XNumberFormatter` converts numbers to strings, or strings to numbers, or detects a number format matching a given string.

```
void attachNumberFormatsSupplier ( [in] com::sun::star::util::XNumberFormatsSupplier xSupplier )
com::sun::star::util::XNumberFormatsSupplier getNumberFormatsSupplier ()

long detectNumberFormat ( [in] long nKey, [in] string aString )

double convertStringToNumber ( [in] long nKey, [in] string aString )
string convertNumberToString ( [in] long nKey, [in] double fValue );

com::sun::star::util::color queryColorForNumber ( [in] long nKey, [in] double fValue,
                                                  [in] com::sun::star::util::color aDefaultColor )
string formatString ( [in] long nKey, [in] string aString );

com::sun::star::util::color queryColorForString ( [in] long nKey, [in] string aString,
                                                  [in] com::sun::star::util::color aDefaultColor )

string getInputString ( [in] long nKey, [in] double fValue )
```

XNumberformatPreviewer

```
string convertNumberToPreviewString ( [in] string aFormat, [in] double fValue,
                                      [in] com::sun::star::lang::Locale nLocale,
                                      [in] boolean bAllowEnglish )

com::sun::star::util::color queryPreviewColorForNumber ( [in] string aFormat, [in] double fValue,
                                      [in] com::sun::star::lang::Locale nLocale,
                                      [in] boolean bAllowEnglish,
                                      [in] com::sun::star::util::color aDefaultColor )
```

This interface `com.sun.star.util.XNumberFormatPreviewer`converts values to strings according to a given format code without inserting the format code into the underlying `com.sun.star.util.NumberFormats` collection.

The example below demonstrates the usage of these interfaces. (OfficeDev/Number_Formats.java)

```
public void doSampleFunction() throws RuntimeException, Exception
{
    // Assume:
    // com.sun.star.sheet.XSpreadsheetDocument maSpreadsheetDoc;
    // com.sun.star.sheet.XSpreadsheet maSheet;

    // Query the number formats supplier of the spreadsheet document
    com.sun.star.util.XNumberFormatsSupplier xNumberFormatsSupplier =
        (com.sun.star.util.XNumberFormatsSupplier)
        UnoRuntime.queryInterface(
        com.sun.star.util.XNumberFormatsSupplier.class, maSpreadsheetDoc );

    // Get the number formats from the supplier
    com.sun.star.util.XNumberFormats xNumberFormats =
        xNumberFormatsSupplier.getNumberFormats();

    // Query the XNumberFormatTypes interface
    com.sun.star.util.XNumberFormatTypes xNumberFormatTypes =
        (com.sun.star.util.XNumberFormatTypes)
        UnoRuntime.queryInterface(
        com.sun.star.util.XNumberFormatTypes.class, xNumberFormats );

    // Get the number format index key of the default currency format,
    // note the empty locale for default locale
    com.sun.star.lang.Locale aLocale = new com.sun.star.lang.Locale();
    int nCurrencyKey = xNumberFormatTypes.getStandardFormat(
        com.sun.star.util.NumberFormat.CURRENCY, aLocale );

    // Get cell range B3:B11
    com.sun.star.table.XCellRange xCellRange =
        maSheet.getCellRangeByPosition( 1, 2, 1, 10 );

    // Query the property set of the cell range
    com.sun.star.beans.XPropertySet xCellProp =
```

```
        (com.sun.star.beans.XPropertySet)
        UnoRuntime.queryInterface(
        com.sun.star.beans.XPropertySet.class, xCellRange );

    // Set number format to default currency
    xCellProp.setPropertyValue( "NumberFormat", new Integer(nCurrencyKey) );

    // Get cell B3
    com.sun.star.table.XCell xCell = maSheet.getCellByPosition( 1, 2 );

    // Query the property set of the cell
    xCellProp = (com.sun.star.beans.XPropertySet)
        UnoRuntime.queryInterface(
        com.sun.star.beans.XPropertySet.class, xCell );

    // Get the number format index key of the cell's properties
    int nIndexKey = ((Integer) xCellProp.getPropertyValue( "NumberFormat" )).intValue();

    // Get the properties of the number format
    com.sun.star.beans.XPropertySet xProp = xNumberFormats.getByKey( nIndexKey );

    // Get the format code string of the number format's properties
    String aFormatCode = (String) xProp.getPropertyValue( "FormatString" );
    System.out.println( "FormatString: `" + aFormatCode + "'" );

    // Create an arbitrary format code
    aFormatCode = "\"wonderful \"" + aFormatCode;

    // Test if it is already present
    nIndexKey = xNumberFormats.queryKey( aFormatCode, aLocale, false );

    // If not, add to number formats collection
    if ( nIndexKey == -1 )
    {
        try
        {
            nIndexKey = xNumberFormats.addNew( aFormatCode, aLocale );
        }
        catch( com.sun.star.util.MalformedNumberFormatException ex )
        {
            System.out.println( "Bad number format code: " + ex );
            nIndexKey = -1;
        }
    }

    // Set the new format at the cell
    if ( nIndexKey != -1 )
        xCellProp.setPropertyValue( "NumberFormat", new Integer(nIndexKey) );
}
```

6.2.6 Document Events

Recurring actions, such as loading, printing or saving, that occur when working with documents, are document *events*, and all documents in OpenOffice.org offer an interface that sends notifications when these events take place.

There are general events common every document, such as loading, printing, or saving, and there are other events that are specific to a particular document type. Both can be accessed through the same interface.

In the document events API, these events are represented by an event name. The following table shows a list of all general document event names:

General Document Event Names	
`OnNew`	New Document was created
`OnLoad`	Document has been loaded
`OnSaveAs`	Document is going to be saved under a new name
`OnSaveAsDone`	Document was saved under a new name
`OnSave`	Document is going to be saved

General Document Event Names	
OnSaveDone	Document was saved
OnPrepareUnload	Document is going to be removed, but still fully available
OnUnload	Document has been removed, document ist still valid, but closing can no longer be prevented
OnFocus	Document was activated
OnUnfocus	Document was deactivated
OnPrint	Document will be printed
OnModifyChange	Modified state of the document has changed

These event names are documented in the `com.sun.star.document.Events` service. Note that this service description exceeds the scope of events that happen on the document as a whole—so it also contains events that can only be accessed by finding the part of the document where the event occurred, for example, a button in a form. This list of events can also be extended by new events, so that future versions of OpenOffice.org can support new types of events through the same API. Therefore, every client that wants to deal with a particular document event must check if this event is supported, or whether it should be prepared to catch an exception.

Every client that is interested in document events can register for being notified. The necessary interface for notification is `com.sun.star.document.XEventBroadcaster`, which is an optional interface of the service`com.sun.star.document.OfficeDocument`. All document objects in OpenOffice.org implement this interface. It has two methods to add and remove listeners for document events:

```
[oneway] void addEventListener( [in] ::com::sun::star::document::XEventListener Listener );
[oneway] void removeEventListener( [in] ::com::sun::star::document::XEventListener Listener );
```

The listeners must implement the interface `com.sun.star.document.XEventListener` and get a notification through a call of their method:

```
[oneway] void notifyEvent( [in] ::com::sun::star::document::EventObject Event );
```

The argument of this call is a `com.sun.star.document.EventObject` struct, which is derived from the usual `com.sun.star.lang.EventObject` and contains two members: the member `Source`, which contains an interface pointer to the event source (here the `com.sun.star.document.OfficeDocument` service) and the member `EventName` which can be one of the names shown in the preceding table.

Both methods in the interface `com.sun.star.document.XEventBroadcaster` can cause problems in scripting languages if the object that implements this interface also implements `com.sun.star.lang.XComponent`, because it has two very similar methods:

```
[oneway] void addEventListener( [in] ::com::sun::star::lang::XEventListener Listener );
[oneway] void removeEventListener( [in] ::com::sun::star::lang::XEventListener Listener );
```

Unfortunately this applies to all OpenOffice.org documents.

In C++ and Java this is no problem, because the complete signature of a method, including the arguments, is used to identify it.

In OpenOffice.org Basic, the fully qualified name including the interface can be used from version 1.1:

```
Sub RegisterListener

  oListener = CreateUnoListener( "DocumentListener_","com.sun.star.document.XEventListener" )

  ThisComponent.com_sun_star_document_XEventBroadcaster_addEventListener( oListener )

End Sub
```

```
Sub DocumentListener_notifyEvent( o as object )

  IF o.EventName = "OnPrepareUnload" THEN
        print o.Source.URL
  ENDIF

end sub

Sub DocumentListener_disposing()
End Sub
```

But the OLE automation bridge, and possibly other scripting language bindings, are unable to distinguish between both `addEventListener()` and `removeEventListener()` methods based on the method signature and must be told which interface you want to use.

You must use the core reflection to get access to either method. The following code shows an example in VBScript, which registers a document event listener at the current document.

```
set xContext = objServiceManager.getPropertyValue( "DefaultContext" )
set xCoreReflection = xContext.getValueByName( "/singletons/com.sun.star.reflection.theCoreReflection" )
set xClass = xCoreReflection.forName( "com.sun.star.document.XEventBroadcaster" )
set xMethod = xClass.getMethod( "addEventListener" )

dim invokeargs(0)
invokeargs(0) = myListener

set value = objServiceManager.Bridge_GetValueObject()
call value.InitInOutParam("[]any", invokeargs)
call xMethod.invoke( objDocument, value )
```

The C++ code below uses OLE Automation. Two helper functions are provided that help to execute UNO operations.

```
// helper function to execute UNO operations via IDispatch
HRESULT ExecuteFunc( IDispatch* idispUnoObject,
                                          OLECHAR* sFuncName,
                                          CComVariant* params,
                                          unsigned int count,
                                          CComVariant* pResult )
{
  if( !idispUnoObject )
    return E_FAIL;

  DISPID id;
  HRESULT hr = idispUnoObject->GetIDsOfNames( IID_NULL, &sFuncName, 1, LOCALE_USER_DEFAULT, &id);
  if( !SUCCEEDED( hr ) ) return hr;

  DISPPARAMS dispparams= { params, 0, count, 0};

  // DEBUG
  EXCEPINFO myInfo;
  return idispUnoObject->Invoke( id, IID_NULL,LOCALE_USER_DEFAULT, DISPATCH_METHOD,
                   &dispparams, pResult, &myInfo, 0);
}

// helper function to execute UNO methods that return interfaces
HRESULT GetIDispByFunc( IDispatch* idispUnoObject,
                                                OLECHAR* sFuncName,
                                                CComVariant* params,
                                                unsigned int count,
                                                CComPtr<IDispatch>& pdispResult )
{
  if( !idispUnoObject )
    return E_FAIL;

  CComVariant result;
  HRESULT hr = ExecuteFunc( idispUnoObject, sFuncName, params, count, &result );
  if( !SUCCEEDED( hr ) ) return hr;

  if( result.vt != VT_DISPATCH || result.pdispVal == NULL )
    return E_FAIL;

  pdispResult = CComPtr<IDispatch>( result.pdispVal );

  return S_OK;
}

// it's assumed that pServiceManager (by creating it as a COM object), pDocument (f.e. by loading it) //
and pListener (the listener we want to add) are passed as parameters

HRESULT AddDocumentEventListener(
 CComPtr<IDispatch> pServiceManager, CComPtr<IDispatch> pDocument, CComPtr<IDispatch> pListener)
```

```
{
  CComPtr<IDispatch> pdispContext;
  hr = GetIDispByFunc( pServiceManager, L"getPropertyValue", &CComVariant( L"DefaultContext" ), 1,
        pdispContext );
  if( !SUCCEEDED( hr ) ) return hr;

  CComPtr<IDispatch> pdispCoreReflection;
  hr = GetIDispByFunc( pdispContext,
                       L"getValueByName",
                       &CcomVariant( L"/singletons/com.sun.star.reflection.theCoreReflection" ),
                       1,
                       pdispCoreReflection );
  if( !SUCCEEDED( hr ) ) return hr;

  CComPtr<IDispatch> pdispClass;
  hr = GetIDispByFunc( pdispCoreReflection,
                       L"forName",
                       &CComVariant( L"com.sun.star.document.XEventBroadcaster" ),
                       1,
                       pdispClass );
  if( !SUCCEEDED( hr ) ) return hr;

  CComPtr<IDispatch> pdispMethod;
  hr = GetIDispByFunc( pdispClass, L"getMethod", &CComVariant( L"addEventListener" ), 1, pdispMethod );
  if( !SUCCEEDED( hr ) ) return hr;

  CComPtr<IDispatch> pdispListener;
  CComPtr<IDispatch> pdispValueObj;
  hr = GetIDispByFunc( mpDispFactory, L"Bridge_GetValueObject", NULL, 0, pdispValueObj );
  if( !SUCCEEDED( hr ) ) return hr;

  CComVariant pValParams[2];
  pValParams[1] = CComVariant( L"com.sun.star.document.XEventListener" );
  pValParams[0] = CComVariant( pdispListener );
  CComVariant dummyResult;
  hr = ExecuteFunc( pdispValueObj, L"Set", pValParams, 2, &dummyResult );
  if( !SUCCEEDED( hr ) ) return hr;

  SAFEARRAY FAR* pPropVal = SafeArrayCreateVector( VT_VARIANT, 0, 1 );
  long ix1 = 0;

  CComVariant aArgs( pdispValueObj );
  SafeArrayPutElement( pPropVal, &ix, &aArgs );

  CComVariant aDoc( pdispDocument );
  CComVariant pParams[2];
  pParams[1] = aDoc;
  pParams[0].vt = VT_ARRAY | VT_VARIANT; pParams[0].parray = pPropVal;

  CComVariant result;

  //invoking the method addeventlistner
  hr = ExecuteFunc( pdispMethod, L"invoke", pParams, 2, &result );
  if( !SUCCEEDED( hr ) ) return hr;

  return S_OK;
}
```

Another way to react to document events is to bind a macro to it—a process called *event binding*. From OpenOffice.org 1.1 you can also use scripts in other languages, provided that a corresponding scripting framework implementation is present.

All document objects in OpenOffice.org support event binding through an interface `com.sun.star.document.XEventsSupplier`. This interface has only one method:

```
::com::sun::star::container::XNameReplace getEvents();
```

This method gives access to a container of event bindings. The container is represented by a `com.sun.star.container.XNameReplace` interface that, together with the methods of its base interfaces, offers the following methods:

```
void replaceByName( [in] string aName, [in] any aElement );
any getByName( [in] string aName );
sequence< string > getElementNames();
boolean hasByName( [in] string aName );
type getElementType();
boolean hasElements();
```

Each container element represents an event binding. By default, all bindings are empty. The element names are the event names shown in the preceding table. In addition, there are document

type-specific events. The method `getElementNames()` yields all possible events that are supported by the object and `hasByName()` checks for the existence of a particular event.

For every supported event name you can use `getByName()` to query for the current event binding or `replaceByName()` to set a new one. Both methods may throw a `com.sun.star.container.NoSuchElementException` exception if an unsupported event name is used.

The type of an event binding, which is wrapped in the `any` returned by `getByName()`, is a sequence of `com.sun.star.beans.PropertyValue` that describes the event binding.

PropertyValue structs in the event binding description	
`EventType`	string. Can assume the values `"StarBasic"` or `"Script"`. The event type `"Script"` describes the location as `URL`. The event type `"StarBasic"` is provided for compatibility reasons and describes the location of the macro through the properties `Library` and `MacroName`, in addition to `URL`.
`Script`	string. Available for the event types `Script` and `StarBasic`. Describes the location of the macro/script routine which is bound. For the URL property, a command URL is expected (see *6.1.6 Office Development - OpenOffice.org Application Environment - Using the Dispatch Framework*). OpenOffice.org will execute this command when the event occurs. For the event type `StarBasic`, the URL uses the *macro:* protocol. For the event type `Script`, other protocols are possible, especially the *script:* protocol. The macro protocol has two forms: *macro:///<Library>.<Module>.<Method(args)>* *macro:///./<Library>.<Module>.<Method(args)>* The first form points to a method in the global basic storage, while the second one points to a method embedded in the current document. *<Library>.<Module>.<Method(args)>* represent the names of the library, the module and the method. Currently, for `args` only string arguments (separated by comma) are possible. If no args exist, empty brackets must be used, because the brackets are part of the scheme. An example URL could look like: *macro:///MyLib.MyModule.MyMethod(foo,bar)* The exact form of the *script:* command URL protocol depends on the installed scripting module. They will be available later as additional components for OpenOffice.org 1.1.
`Library`	string. Deprecated. Available for `EventType "StarBasic"`. Can assume the values `"application"` or empty string for the global basic storage, and `"document"` for the document where the code is embedded.
`MacroName`	string. Deprecated. Available for `EventType "StarBasic"`. Describes the macro location as `<Library>.<MyModule>.<MyMethod>`.



In OpenOffice.org 1.1 all properties (`URL`, `Library`, `MacroName`) will be returned for event bindings of type `StarBasic`, regardless if the binding was created with a `URL` property only or with the `Library` and `MacroName` property. The internal implementation does the necessary conversion. Older versions of OpenOffice.org always returned only `Library` and `MacroName`, even if the binding was created with the `URL` property.

In OpenOffice.org 1.1 there is another important extension in the area of document events and event bindings. This version has a new service `com.sun.star.document.GlobalEventBroadcaster` that offers the same document-event-related functionality as described previously (interfaces `com.sun.star.document.XEventBroadcaster`,

com.sun.star.document.XEventsSupplier), but it allows you to register for events that happen in any document and also allows you to set bindings for all documents that are stored in the global UI configuration of OpenOffice.org. Using this services frees you from registering at every single document that has been created or loaded.

Though a potential listener registers for event notifications at this global service and not at any document itself, the received event source in the event notification is the *document,* not the GlobalEventBroadcaster. The reason for this is that usually a listener contains code that works on the document, so it needs a reference to it.

The service com.sun.star.document.GlobalEventBroadcaster also supports two more events that do not occur in any document but are useful for working with document events:

Global Event Names	
OnStartApp	Application has been started
OnCloseApp	Application is going to be closed. This event is fired after all documents have been closed and nobody objected to the shutdown.

The event source in the notifications is NULL (empty).

All event bindings can be seen or set in the OpenOffice.org UI in the **Tools-Configure** dialog on **Events** page. Two radio buttons on the right side of the dialog toggle between **OpenOffice.org** and **Document** binding. In OpenOffice.org 1.1, you can still only bind to OpenOffice.org Basic macros in the dialog. Bindings to *script:* URLs can only be set using the API, but the dialog is at least able to display them. If, in OpenOffice.org 1.1, a global and a document binding are set for the same event, first the global and then the document binding is executed. With older versions, only the document binding was executed, and the global binding was only executed if no document binding was set.

6.2.7 Path Organization

The path settings service is the central service that manages the paths of OpenOffice.org. Almost every component inside OpenOffice.org uses one or more of the paths to access its resources located on the file system.

Users can customize most of the paths in OpenOffice.org by choosing **Tools – Options – OpenOffice.org – Paths**.

Path Settings

The com.sun.star.util.PathSettings service supports a number of properties which store the OpenOffice.org predefined paths. There are two different groups of properties. One group stores only a single path and the other group stores two or more paths - separated by a semicolon.

Properties of com.sun.star.util.PathSettings		
Addin	Single path	Specifies the directory that contains spreadsheet add-ins which use the old add-in API.
AutoCorrect	Multi path	Specifies the directories that contain the settings for the AutoCorrect dialog.
AutoText	Multi path	Specifies the directories that contain the AutoText modules.
Backup	Single path	Specifies the directory for storing automatic backup copies of documents.

Properties of com.sun.star.util.PathSettings		
Basic	Multi path	Specifies the location of the Basic files that are used by the AutoPilots.
Bitmap	Single path	Specifies the directory that contains the external icons for the toolbars.
Config	Single path	Specifies the location of the configuration files. This property is not visible in the OpenOffice.org path options dialog and cannot be changed by users.
Dictionary	Single path	Specifies the location of the OpenOffice.org dictionaries.
Favorite	Single path	Specifies the directory that contains the saved folder bookmarks.
Filter	Single path	Specifies the directory where the filters are stored.
Gallery	Multi path	Specifies the directories that contain the Gallery database and multimedia files.
Graphic	Single path	Specifies the directory that is displayed when the dialog for opening a graphic or for saving a new graphic is called.
Help	Single path	Specifies the location of the Office help files.
Linguistic	Single path	Specifies the directory where the spellcheck files are stored.
Module	Single path	Specifies the directory where the modules are stored.
Palette	Single path	Specifies the location of the palette files that contain user-defined colors and patterns (*.SOB and *.SOF).
Plugin	Multi path	Specifies the directories where the Plugins are stored.
Storage	Single path	Specifies the directory where mail and news files as well as other information (for example, about FTP Server) are stored. This property is not visible in the OpenOffice.org path options dialog and cannot be changed by users.
Temp	Single path	Specifies the directory for the office temp-files.
Template	Multi path	Specifies the directory for the OpenOffice.org document templates.
UIConfig	Multi path	Specifies the location of global directories when looking for user interface configuration files. The user interface configuration is merged with the user settings that are stored in the directory specified by *UserConfig*.
UserConfig	Single path	Specifies the directory that contains the user settings, including the user interface configuration files for menus, toolbars, accelerators and status bars.
UserDictionary	Single path	Specifies the directory for the custom dictionaries.
Work	Single path	Specifies the location of the work folder. This path can be modified according to the user's needs and can be seen in the Open or Save dialog.

Configuration

The path settings service uses the group `Path` in the *org.Openoffice.Office.Common* branch to read and store paths. The `Current` and `Default` groups in the share layer of the configuration branch store the path settings properties. The `Current` group initialize the properties of the path settings service during startup. If the user activates the **Default** button in the path options dialog, the `Default` group values are copied to the current ones.



Note: The configuration branch separates the paths of a property with a colon (:), whereas the path settings service separates multiple paths with a semicolon (;).

```
<?xml version='1.0' encoding='UTF-8'?>
<oor:component-schema oor:name="Common" oor:package="org.openoffice.Office" xml:lang="en-US"
xmlns:oor="http://openoffice.org/2001/registry" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://wwww.w3.org/2001/XMLSchema-instance">
    <component>
        <group oor:name="Path">
            <group oor:name="Current">
                <prop oor:name="OfficeInstall" oor:type="xs:string">
                    <value/>
                </prop>
                <prop oor:name="OfficeInstallURL" oor:type="xs:string">
                    <value/>
                </prop>
                <prop oor:name="Addin" oor:type="xs:string">
                    <value>$(progpath)/addin</value>
                </prop>
                <prop oor:name="AutoCorrect" oor:type="oor:string-list">
                    <value oor:separator=":">$(insturl)/share/autocorr:$(userurl)/autocorr</value>
                </prop>
                <prop oor:name="AutoText" oor:type="oor:string-list">
                    <value oor:separator=":">
                        $(insturl)/share/autotext/$(vlang):$(userurl)/autotext
                    </value>
                </prop>
                <prop oor:name="Backup" oor:type="xs:string">
                    <value>$(userurl)/backup</value>
                </prop>
                <prop oor:name="Basic" oor:type="oor:string-list">
                    <value oor:separator=":">$(insturl)/share/basic:$(userurl)/basic</value>
                </prop>
                <prop oor:name="Bitmap" oor:type="xs:string">
                    <value>$(insturl)/share/config/symbol</value>
                </prop>
                <prop oor:name="Config" oor:type="xs:string">
                    <value>$(insturl)/share/config</value>
                </prop>
                <prop oor:name="Dictionary" oor:type="xs:string">
                    <value>$(insturl)/share/wordbook/$(vlang)</value>
                </prop>
                <prop oor:name="Favorite" oor:type="xs:string">
                    <value>$(userurl)/config/folders</value>
                </prop>
                <prop oor:name="Filter" oor:type="xs:string">
                    <value>$(progpath)/filter</value>
                </prop>
                <prop oor:name="Gallery" oor:type="oor:string-list">
                    <value oor:separator=":">$(insturl)/share/gallery:$(userurl)/gallery</value>
                </prop>
                <prop oor:name="Graphic" oor:type="xs:string">
                    <value>$(insturl)/share/gallery</value>
                </prop>
                <prop oor:name="Help" oor:type="xs:string">
                    <value>$(instpath)/help</value>
                </prop>
                <prop oor:name="Linguistic" oor:type="xs:string">
                    <value>$(insturl)/share/dict</value>
                </prop>
                <prop oor:name="Module" oor:type="xs:string">
                    <value>$(progpath)</value>
                </prop>
                <prop oor:name="Palette" oor:type="xs:string">
                    <value>$(userurl)/config</value>
                </prop>
                <prop oor:name="Plugin" oor:type="oor:string-list">
                    <value oor:separator=":">$(userpath)/plugin</value>
                </prop>
                <prop oor:name="Storage" oor:type="xs:string">
                    <value>$(userpath)/store</value>
                </prop>
                <prop oor:name="Temp" oor:type="xs:string">
                    <value>$(temp)</value>
                </prop>
                <prop oor:name="Template" oor:type="oor:string-list">
                    <value oor:separator=":">
                        $(insturl)/share/template/$(vlang):$(userurl)/template
                    </value>
                </prop>
                <prop oor:name="UIConfig" oor:type="oor:string-list">
                    <value oor:separator=":"/>
                </prop>
                <prop oor:name="UserConfig" oor:type="xs:string">
                    <value>$(userurl)/config</value>
```

```
            </prop>
            <prop oor:name="UserDictionary" oor:type="xs:string">
                <value>$(userurl)/wordbook</value>
            </prop>
            <prop oor:name="Work" oor:type="xs:string">
                <value>$(work)</value>
            </prop>
        </group>
        <group oor:name="Default">
            <prop oor:name="Addin" oor:type="xs:string">
                <value>$(progpath)/addin</value>
            </prop>
            <prop oor:name="AutoCorrect" oor:type="oor:string-list">
                <value oor:separator=":">
                    $(insturl)/share/autocorr:$(userurl)/autocorr
                </value>
            </prop>
            <prop oor:name="AutoText" oor:type="oor:string-list">
                <value oor:separator=":">
                    $(insturl)/share/autotext/$(vlang):$(userurl)/autotext
                </value>
            </prop>
            <prop oor:name="Backup" oor:type="xs:string">
                <value>$(userurl)/backup</value>
            </prop>
            <prop oor:name="Basic" oor:type="oor:string-list">
                <value oor:separator=":">$(insturl)/share/basic:$(userurl)/basic</value>
            </prop>
            <prop oor:name="Bitmap" oor:type="xs:string">
                <value>$(insturl)/share/config/symbol</value>
            </prop>
            <prop oor:name="Config" oor:type="xs:string">
                <value>$(insturl)/share/config</value>
            </prop>
            <prop oor:name="Dictionary" oor:type="xs:string">
                <value>$(insturl)/share/wordbook/$(vlang)</value>
            </prop>
            <prop oor:name="Favorite" oor:type="xs:string">
                <value>$(userurl)/config/folders</value>
            </prop>
            <prop oor:name="Filter" oor:type="xs:string">
                <value>$(progpath)/filter</value>
            </prop>
            <prop oor:name="Gallery" oor:type="oor:string-list">
                <value oor:separator=":">$(insturl)/share/gallery:$(userurl)/gallery</value>
            </prop>
            <prop oor:name="Graphic" oor:type="xs:string">
                <value>$(insturl)/share/gallery</value>
            </prop>
            <prop oor:name="Help" oor:type="xs:string">
                <value>$(instpath)/help</value>
            </prop>
            <prop oor:name="Linguistic" oor:type="xs:string">
                <value>$(insturl)/share/dict</value>
            </prop>
            <prop oor:name="Module" oor:type="xs:string">
                <value>$(progpath)</value>
            </prop>
            <prop oor:name="Palette" oor:type="xs:string">
                <value>$(userurl)/config</value>
            </prop>
            <prop oor:name="Plugin" oor:type="oor:string-list">
                <value oor:separator=":">$(userpath)/plugin</value>
            </prop>
            <prop oor:name="Temp" oor:type="xs:string">
                <value>$(temp)</value>
            </prop>
            <prop oor:name="Template" oor:type="oor:string-list">
                <value oor:separator=":">
                    $(insturl)/share/template/$(vlang):$(userurl)/template
                </value>
            </prop>
            <prop oor:name="UIConfig" oor:type="oor:string-list">
                <value oor:separator=":"/>
            </prop>
            <prop oor:name="UserConfig" oor:type="xs:string">
                <value>$(userurl)/config</value>
            </prop>
            <prop oor:name="UserDictionary" oor:type="xs:string">
                <value>$(userurl)/wordbook</value>
            </prop>
            <prop oor:name="Work" oor:type="xs:string">
                <value>$(work)</value>
            </prop>
        </group>
    </group>
</component>
```

```
</oor:component-schema>
```

Accessing Path Settings

The path settings service is a one-instance service that supports the `com.sun.star.beans.XPropertySet`, `com.sun.star.beans.XFastPropertySet` and `com.sun.star.beans.XMultiPropertySet` interfaces for access to the properties.

The service can be created using the service manager of OpenOffice.org and the service name *com.sun.star.util.PathSettings*. The following example creates the path settings service.

```
import com.sun.star.lang.XMultiServiceFactory;
import com.sun.star.uno.Exception;
import com.sun.star.uno.XInterface;
import com.sun.star.beans.XPropertySet

XPropertySet createPathSettings() {

    // Obtain Process Service Manager.
    XMultiServiceFactory xServiceFactory = ...

    // Create Path settings service. Needs to be done only once per process.
    XInterface xPathSettings;
    try {
        xPathSettings  = xServiceFactory.createInstance("com.sun.star.util.PathSettings" );
    }
    catch (com.sun.star.uno.Exception e) {
    }

    if (xPathSettings  != null)
        return (XpropertySet) UnoRuntime.queryInterface(XPropertySet.class, xPathSettings);
    else
        return null;
}
```

The main interface of the path settings service is `com.sun.star.beans.XPropertySet`. You can retrieve and write properties with this interface. It also supports getting information about the properties themselves.

- `com::sun::star::beans::XPropertySetInfo getPropertySetInfo();`
  The path settings service returns an `XPropertySetInfo` interface where more information about the path properties can be retrieved. The information includes the name of the property, a handle for faster access with `XFastPropertySet`, the type of the property value and attribute values.
- `void setPropertyValue( [in] string aPropertyName, [in] any aValue );`
  This function can set the path properties to a new value. The path settings service expects that a value of type `string` is provided. The string must be a correctly encoded file URL. If the path property supports multiple paths, each path must be separated by a semicolon (;). Path variables are also allowed, so long as they can be resolved to a valid file URL.
- `any getPropertyValue( [in] string PropertyName );`
  This function retrieves the value of a path property. The property name must be provided and the path is returned. The path settings service always returns the path as a file URL. If the property value includes multiple paths, each path is separated by a semicolon (;).



Note: The path settings service always provides property values as file URLs. Properties which are marked as multi path (see table above) use a semicolon (;) as a separator for the different paths. The service also expects that a new value for a path property is provided as a file URL or has a preceding path variable, otherwise a com.sun.star.lang.IllegalArgumentException is thrown.

Illustration 54 shows how the path settings, path substitution, and configuration service work together to read or write path properties.

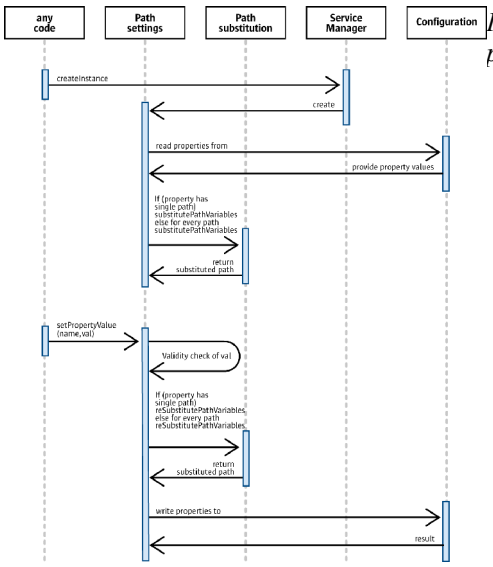


Illustration 68: Interaction of path settings, path substitution and configuration



Important: Keep in mind that the paths managed by the path settings service are vital for almost all of the functions in OpenOffice.org. Entering a wrong path can result in minor malfunctions or break the complete OpenOffice.org installation. Although the path settings service performs a validity check on the provided URL, this cannot prevent all problems.

The following code example uses the path settings service to retrieve and set the path properties.

```
import com.sun.star.bridge.XUnoUrlResolver;
import com.sun.star.uno.UnoRuntime;
import com.sun.star.uno.XComponentContext;
import com.sun.star.lang.XMultiComponentFactory;
import com.sun.star.beans.XPropertySet;
import com.sun.star.beans.PropertyValue;

import com.sun.star.beans.UnknownPropertyException;

/* Provides example code how to access and use the
 * path pathsettings servce.
 */
public class PathSettingsTest extends java.lang.Object {

   /*
    * List of predefined path variables supported by
    * the path settings service.
    */
   private static String[] predefinedPathProperties = {
      "Addin",
      "AutoCorrect",
      "AutoText",
      "Backup",
      "Basic",
      "Bitmap",
      "Config",
      "Dictionary",
      "Favorite",
      "Filter",
      "Gallery",
      "Graphic",
      "Help",
      "Linguistic",
      "Module",
      "Palette",
      "Plugin",
      "Storage",
      "Temp",
      "Template",
      "UIConfig",
      "UserConfig",
      "UserDictionary",
      "Work"
   };

   /*
    * @param args the command line arguments
```

```
    */
   public static void main(String[] args) {

      XComponentContext xRemoteContext = null;
      XMultiComponentFactory xRemoteServiceManager = null;
      XPropertySet xPathSettingsService = null;

      try {
         // connect
         XComponentContext xLocalContext =
            com.sun.star.comp.helper.Bootstrap.createInitialComponentContext(null);
         XMultiComponentFactory xLocalServiceManager = xLocalContext.getServiceManager();
         Object urlResolver  = xLocalServiceManager.createInstanceWithContext(
            "com.sun.star.bridge.UnoUrlResolver", xLocalContext );
         XUnoUrlResolver xUnoUrlResolver = (XUnoUrlResolver) UnoRuntime.queryInterface(
            XUnoUrlResolver.class, urlResolver );
         Object initialObject = xUnoUrlResolver.resolve(
            "uno:socket,host=localhost,port=8100;urp;StarOffice.ServiceManager" );
         XPropertySet xPropertySet = (XPropertySet)UnoRuntime.queryInterface(
            XPropertySet.class, initialObject);
         Object context = xPropertySet.getPropertyValue("DefaultContext");
         xRemoteContext = (XComponentContext)UnoRuntime.queryInterface(
            XComponentContext.class, context);
         xRemoteServiceManager = xRemoteContext.getServiceManager();

         Object pathSubst = xRemoteServiceManager.createInstanceWithContext(
            "com.sun.star.comp.framework.PathSettings", xRemoteContext );
         xPathSettingsService = (XPropertySet)UnoRuntime.queryInterface(
            XPropertySet.class, pathSubst);

         /* Work with path settings */
         workWithPathSettings( xPathSettingsService );
      }
      catch (java.lang.Exception e){
         e.printStackTrace();
      }
      finally {
         System.exit(0);
      }
   }

   /*
    * Retrieve and set path properties from path settings service
    * @param xPathSettingsService the path settings service
    */
   public static void workWithPathSettings(XPropertySet xPathSettingsService) {
      if (xPathSettingsService != null) {
         for (int i=0; i<predefinedPathProperties.length; i++) {
            try {
               /* Retrieve values for path properties from path settings service*/
               Object aValue = xPathSettingsService.getPropertyValue(predefinedPathProperties[i]);

               // getPropertyValue returns an Object, you have to cast it to type that you need
               String aPath = (String)aValue;
               System.out.println("Property="+ predefinedPathProperties[i] + " Path=" + aPath);
            }
            catch (com.sun.star.beans.UnknownPropertyException e) {
               System.out.println("UnknownPropertyException has been thrown accessing
"+predefinedPathProperties[i]);
            }
            catch (com.sun.star.lang.WrappedTargetException e) {
               System.out.println("WrappedTargetException has been thrown accessing
"+predefinedPathProperties[i]);
            }
         }

         // Try to modfiy the work path property. After running this example
         // you should see the new value of "My Documents" in the path options
         // tab page, accessible via "Tools - Options - OpenOffice.org - Paths".
         // If you want to revert the changes, you can also do it with the path tab page.
         try {
            xPathSettingsService.setPropertyValue("Work", "$(temp)");
            String aValue = (String)xPathSettingsService.getPropertyValue("Work");
            System.out.println("The work path should now be " + aValue);
         }
         catch (com.sun.star.beans.UnknownPropertyException e) {
            System.out.println("UnknownPropertyException has been thrown accessing PathSettings
service");
         }
         catch (com.sun.star.lang.WrappedTargetException e) {
            System.out.println("WrappedTargetException has been thrown accessing PathSettings service");
         }
         catch (com.sun.star.beans.PropertyVetoException e) {
            System.out.println("PropertyVetoException has been thrown accessing PathSettings service");
         }
         catch (com.sun.star.lang.IllegalArgumentException e) {
```

```
            System.out.println("IllegalArgumentException has been thrown accessing PathSettings
service");
          }
       }
    }
}
```

Path Variables

Path variables are used as placeholders for system-dependent paths or parts of paths which are only known during the runtime of OpenOffice.org. The path substitution service `com.sun.star.util.PathSubstitution` - which manages all path variables of OpenOffice.org - checks the runtime environment during startup and sets the values of the path variables. The path substitution service supports a number of predefined path variables. They provide information about important paths that OpenOffice.org currently uses. They are implemented as read-only values and cannot be changed.

OpenOffice.org is a multi-platform solution that runs on different file systems. Obviously users want to have a single user configuration on all workstations across all platforms in a networked installation. For example, a user wants to use both the Windows and Unix version of OpenOffice.org. The home directory and the working directory are located on a central file server that uses Samba to provide access for Windows systems. The user only wants to have one user installation for both systems, so that individual settings only need to be specified once.

The path settings service described in *6.2.11 Office Development - Common Application Features - Path Organization - Path Settings* utilizes the path substitution service. In the configuration of OpenOffice.org, path variables describe the path settings, and these variables can be substituted by platform-specific paths during startup. That way, path substitution gives users the power to apply path settings only once, while the system takes care of the necessary platform-dependent and environment adaptations.

Illustration 53 shows how a path variable can resolve the path problem that arises when you use the same user directory on different platforms.

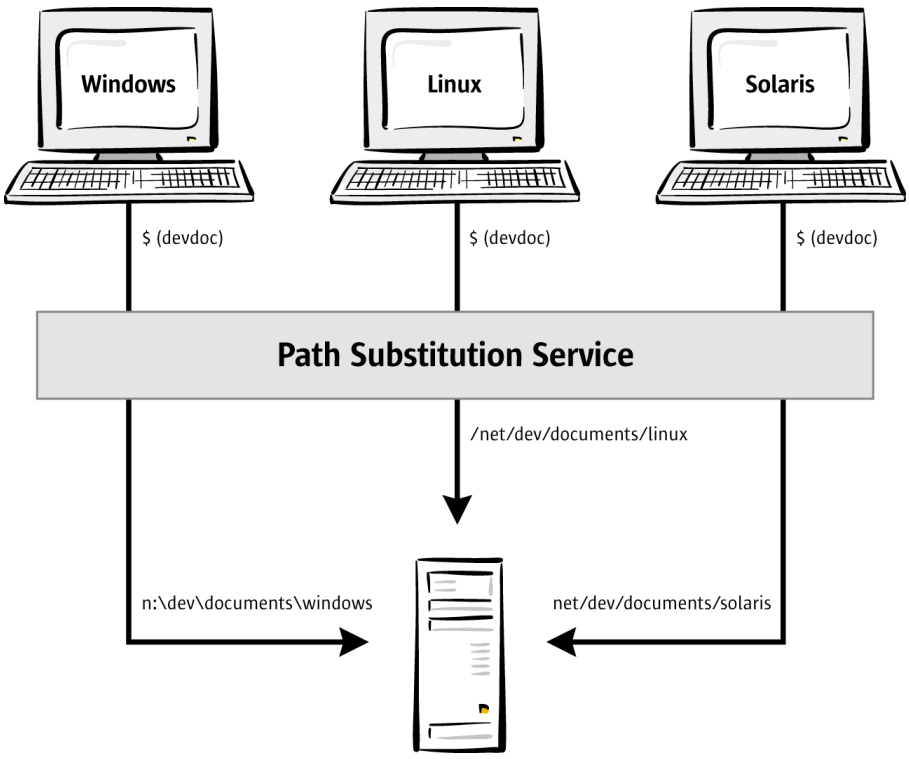


Illustration 69: Path variables solve problems in heterogeneous environments

The following sections describe predefined variables, how to define your own variables, and how to resolve path variables with respect to paths in your programs.

Predefined Variables

The path substitution service supports a number of predefined path variables. They provide information about the paths that OpenOffice.org currently uses. They are implemented as read-only values and cannot be modified.

The predefined path variables can be separated into three distinct groups. The first group of variables specifies a *single path*, the second group specifies a *list of paths* that are separated by the shell or operating system dependent character, and the third group specifies only a *part of a path.*

All predefined variable names are case insensitive, as opposed to the user-defined variables that are described below.

Predefined variables supported by service com.sun.star.util.PathSubstitution		
`$(home)`	Single path	The absolute path to the home directory of the current user. Under Windows this depends on the specific versions: usually the *<drive>:\Documents and Settings\<username>\Application Data* under Windows 2000/XP and *<drive>:\Windows\Profiles\<username>\Application Data* under Windows NT and Win9x, ME with multi user support. Windows 9x and ME without multi-user support *<drive>:\Windows\Application Data.*
`$(inst)` `$(instpath)` `$(insturl)`	Single path	The absolute installation path of OpenOffice.org. Normally the *share* and *program* folders are located inside the installation folder. The `$(instpath)` and `$(insturl)` variables are aliases to `$(inst)`—they are included for downward compatibility and should not be used.

Predefined variables supported by service com.sun.star.util.PathSubstitution		
$(prog) $(progpath) $(progurl)	Single path	The absolute path of the program folder of OpenOffice.org. Normally the executable and the shared libraries are located in this folder. The $(prog-path) and $(progurl) variables are aliases to $(prog)—they are supported for downward compatibility and should not be used.
$(temp)	Single path	The absolute path of the current temporary directory used by OpenOffice.org.
$(user) $(userpath) $(userurl)	Single path	The absolute path to the user installation folder of OpenOffice.org. The $(userpath) and $(userurl) variables are aliases to $(user)—they are supported for downward compatibility and should not be used.
$(work)	Single path	The absolute path of the working directory of the user. Under Windows this is the *My Documents* folder. Under Unix this is the *home* directory of the user.
$(path)	List of paths	The value of the PATH environment variable of the OpenOffice.org process. The single paths are separated by a ';' character independent of the system.
$(lang)	Part of a path	The country code used by OpenOffice.org, see the table Mapping ISO 639/3166 to $(lang) below for examples.
$(langid)	Part of a path	The language identifier used by OpenOffice.org. An identifier is composed of a primary language identifier and a sublanguage identifier such as 0x0009=English (primary language identifier), 0x0409=English US (composed language code).The language identifier is based on the Microsoft language identifiers, for further information please see: Table of Language Identifiers http://msdn.microsoft.com/library/en-us/intl/nls_238z.asp Primary Language Identifiers http://msdn.microsoft.com/library/en-us/intl/nls_61df.asp SubLanguage Identifiers http://msdn.microsoft.com/library/en-us/intl/nls_19ir.asp
$(vlang)	Part of a path	The language used by OpenOffice.org as an English string, for example, "german" for a German version of OpenOffice.org.

The values of $(lang), $(langid) and $(vlang) are based on the property *ooLocale* in the configuration branch *org.openoffice.Setup/L10N*, that is normally located in the *share* directory. This property follows the ISO 639-1/ISO3166 standards that define identification codes for languages and countries. The *ooLocale* property is written by the setup application during installation time. The following are examples of table Mapping ISO 639/3166 to $(vlang):

Mapping from ISO639-1/ISO3166 to $(lang) and $(vlang)			
ISO 639-1	**ISO 3166**	**$(lang)**	**$(vlang)**
ar	*	96	arabic
ca	AD	37	catalan
ca	ES	37	catalan
cs	*	42	czech
cz	*	42	czech
da	DK	45	danish
de	*	49	german
el	*	30	greek
en	*	1	english

Mapping from ISO639-1/ISO3166 to $(lang) and $(vlang)			
en	GB	1	english_uk
es	*	34	spanish
fi	FI	35	finnish
fr	*	33	french
he	*	97	hebrew
hu	HU	36	hungarian
it	*	39	italian
ja	JP	81	japanese
ko	*	82	korean
nb	NO	47	norwegian
nl	*	31	dutch
nn	NO	47	norwegian
no	NO	47	norwegian
pl	PL	48	polish
pt	BR	55	portuguese_brazilian
pt	PT	3	portuguese
ru	RU	7	russian
sk	SK	43	slovak
sv	*	46	swedish
th	TH	66	thai
tr	TR	90	turkish
zh	CN	86	chinese_simplified
zh	TW	88	chinese_traditional

Custom Path Variables

Syntax

The path substitution service supports the definition and usage of user-defined path variables. The variable names must use this syntax:

```
variable ::= "$(" letter { letter | digit } ")"
letter ::= "A"-"Z"|"a"-"z"
digit ::= "0"-"9"
```

The user-defined variables must be defined in the configuration branch *org.openoffice.Office.Substitution.* OpenOffice.org employs a rule-based system to evaluate which definition of a user-defined variable is chosen. The following sections describe the different parts of this rule-based system and the configuration settings that are required for defining new path variables.

Environment Values

To bind a specific value to a user-defined path variable, the path substitution service uses environment values. The path substitution service chooses a variable definition based on the values of these environment parameters. The following table describes which parameters can be used:

Environment parameters	
`Host`	This value can be a host name or an IP address , depending on the network configuration (DNS server available). A host name is case insensitive and can also use the asterisk (*) wildcard to represent match zero or more characters.
`YPDomain`	The yellow pages domain or NIS domain. The value is case insensitive and can use the asterisk (*) wildcard to represent match zero or more characters.
`DNSDomain`	The domain name service. The value is case insensitive and can use the asterisk (*) wildcard to represent match zero or more characters.
`NTDomain`	Windows NT domain. The value is case insensitive and can use the asterisk (*) wildcard to represent match zero or more characters.
`OS`	The operating system parameter supports the following values: • WINDOWS (all windows versions including Win9x, WinME, and WinXP) • UNIX (includes LINUX and SOLARIS) • SOLARIS • LINUX

Rules

The user can define the mapping of environment parameter values to variable values. Each definition is called a *rule* and all rules for a particular variable are the *rule set*. You can only have one environment parameter value for each rule.

The following example rules specify that the user-defined variable called `devdoc` is bound to the directory *s:\develop\documentation* if OpenOffice.org is running under Windows. The second rule binds `devdoc` to */net/develop/documentation* if OpenOffice.org is running under Solaris.

```
Variable name=devdoc
Environment parameter=OS
Value=file:///s:/develop/documentation

Variable name=devdoc
Environment parameter=SOLARIS
Value=file:///net/develop/documentation
```

Analyzing User-Defined Rules

OpenOffice.org uses matching rules to find the active rule inside a provided rule set.

1. Tries to match with the `Host` environment parameter. If more than one rule matches—this can be possible if you use the asterisk (*) wildcard character - the first matching rule is applied.
2. Tries to match with the different `Domain` parameters. There is no predefined order for the domain parameters - the first matching rule is applied.
3. Try to match with the `OS` parameter. The specialized values have a higher priority than generic ones, for example, LINUX has a higher priority than UNIX.

Illustration 51 shows the analyzing and matching of user-defined rules.

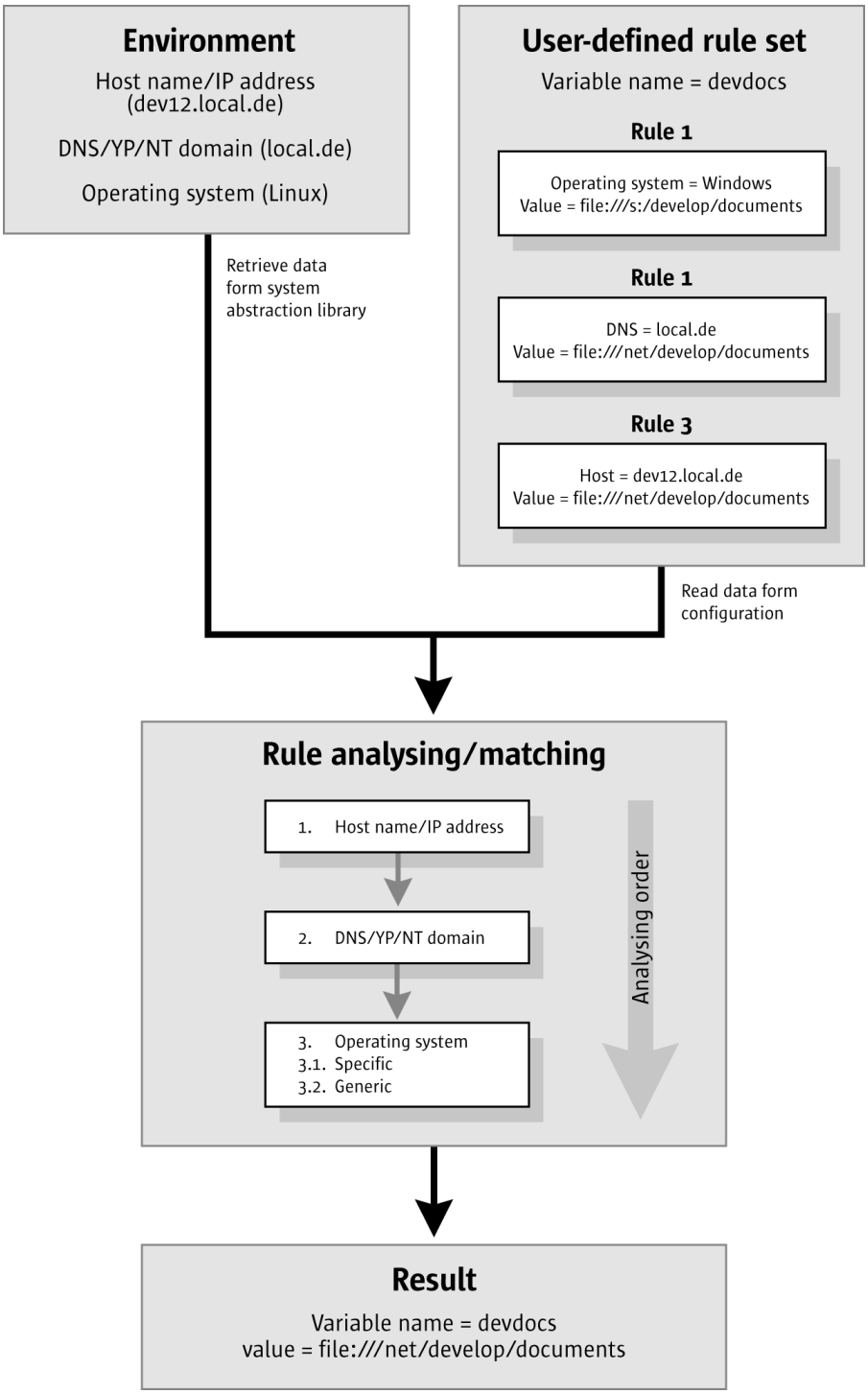


Illustration 70: Process of the rule set analyzing

The analyzing and matching process is done whenever a rule set has changed. Afterwards the values of the user-defined path variables are set and can be retrieved using the interface `com.sun.star.util.XStringSubstitution`.

Configuration

The path substitution service uses the *org.openoffice.Office.Substitution* configuration branch for the rule set definitions, which adhere to this schema:

```
<?xml version='1.0' encoding='UTF-8'?>
<oor:component-schema oor:name="Substitution" oor:package="org.openoffice.Office" xml:lang="en-US"
xmlns:oor="http://openoffice.org/2001/registry" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <templates>
        <group oor:name="SharePointMapping">
            <prop oor:name="Directory" oor:type="xs:string" oor:nillable="false"/>
            <group oor:name="Environment">
                <prop oor:name="OS" oor:type="xs:string"/>
                <prop oor:name="Host" oor:type="xs:string"/>
                <prop oor:name="DNSDomain" oor:type="xs:string"/>
                <prop oor:name="YPDomain" oor:type="xs:string"/>
                <prop oor:name="NTDomain" oor:type="xs:string"/>
            </group>
        </group>
        <set oor:name="SharePoint" oor:node-type="SharePointMapping"/>
    </templates>
    <component>
        <set oor:name="SharePoints" oor:node-type="SharePoint"/>
    </component>
</oor:component-schema>
```

The `SharePoints` set is the root container that store the definition of the different user-defined path variables. The `SharePoint` set uses nodes of type `SharePoint` which defines a single user-defined path variable.

Properties of the SharePoint set nodes	
`oor:component-data`	String. The name of the user-defined path variable. It must be unique inside the *SharePoints* set. The name must meet the requirements for path variable names, see *6.2.11 Office Development - Common Application Features - Path Organization - Path Variables* The preceding characters "`$(`" and the succeeding "`)`" must be omitted, for example, the node string for the path variable `$(devdoc)` must be `devdoc`.

A `SharePoint` set is a container for the different rules, called `SharePointMapping` in the configuration.

Properties of the SharePointMapping group		
`oor:component-data`	String - must be unique inside the *SharePoint* set, but with no additional meaning for user-defined path variables. Use a consecutive numbering scheme - even numbers are permitted.	
`Directory`	String - must be set and contain a valid and encoded file URL that represents the value of the user-defined path variable for the rule.	
`Environment`	Group - contains a set of properties that define the environment parameter that this rule must match. You can only use *one* environment in a rule.	
	`OS`	The operating system. The following values are supported: • WINDOWS = Matches all Windows OS from Win 98 and higher. • LINUX = Matches all supported Linux systems. • SOLARIS = Matches all supported Solaris systems. • UNIX = Matches all supported Unix systems (Linux,Solaris)
	`Host`	The host name or IP address. The name or address can include the asterisk (*) wildcard to match with zero or more characters. For example, *dev*.local.de* refers to all systems where the host name starts with "dev" and ends with ".local.de"

Properties of the SharePointMapping group		
	DNSDomain	The domain name service. The value is case insensitive and can use the asterisk (*) wildcard for zero or more characters.
	YPDomain	The yellow pages domain or NIS domain. The value is case insensitive and can use the asterisk (*) wildcard for zero or more characters.
	NTDomain	Windows NT domain. The value is case insensitive and can use the asterisk (*) wildcard for zero or more characters.

The following example uses two rules to map a Windows and Unix specific path to the user-defined path variable *MyDocuments*.

```
<?xml version="1.0" encoding="utf-8"?>
<oor:component-data oor:name="Substitution" oor:context="org.openoffice.Office"
xsi:schemaLocation="http://openoffice.org/2001/registry component-update.xsd"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:oor="http://openoffice.org/2001/registry"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <node oor:name="SharePoints">
        <node oor:name="MyDocuments" oor:op="replace">
            <node oor:name="1" oor:op="replace">
                <prop oor:name="Directory"><value>file:///H:/documents</value></prop>
                <node oor:name="Environment">
                    <prop oor:name="OS"><value>Windows</value></prop>
                </node>
            </node>
            <node oor:name="2" oor:op="replace">
                <prop oor:name="Directory"><value>file:///net/home/user/documents</value></prop>
                <node oor:name="Environment">
                    <prop oor:name="OS"><value>UNIX</value></prop>
                </node>
            </node>
        </node>
    </node>
</oor:component-data>
```

Resolving Path Variables

This section explains how to use the OpenOffice.org implementation of the path substitution service. The following code snippet creates a path substitution service.

```
import com.sun.star.lang.XMultiServiceFactory;
import com.sun.star.uno.Exception;
import com.sun.star.uno.XInterface;
import com.sun.star.util.XStringSubstitution

XStringSubstitution createPathSubstitution() {

    ////////////////////////////////////////////////////////////////////////
    // Obtain Process Service Manager.
    ////////////////////////////////////////////////////////////////////////

    XMultiServiceFactory xServiceFactory = ...

    ////////////////////////////////////////////////////////////////////////
    // Create Path Substitution. This needs to be done only once per process.
    ////////////////////////////////////////////////////////////////////////

    XInterface xPathSubst;
    try {
        xPathSubst = xServiceFactory.createInstance(
           "com.sun.star.util.PathSubstitution" );
    }
    catch (com.sun.star.uno.Exception e) {
    }

    if (xPathSubst != null)
        return (XStringSubstitution)UnoRuntime.queryInterface(
            XStringSubstitution.class, xPathSubst );
    else
        return null;
}
```

The service is implemented as a one-instance service and supports the interface `com.sun.star.util.XStringSubstitution`. The interface has three methods:

```
string substituteVariables( [in] string aText, [in] boolean bSubstRequired )
string reSubstituteVariables( [in] string aText )
string getSubstituteVariableValue( [in] string variable )
```

The method `substituteVariables()` returns a string where all known variables are replaced by their value. Unknown variables are not replaced. The argument `bSubstRequired` can be used to indicate that the client needs a full substitution—otherwise the function fails and throws a `com.sun.star.container.NoSuchElementException`. For example: *$(inst)/share/autotext/$(vlang)* could be substituted to *file:///c:/OpenOffice.org1.0.2/share/autotext/english*.

The method `reSubstituteVariables()` returns a string where parts of the provided path `aText` are replaced by variables that represent this part of the path. If a matching variable is not found, the path is not modified.

The predefined variable `$(path)` is not used for substitution. Instead, it is a placeholder for the path environment variable does not have a static value during runtime. The path variables `$(lang)`, `$(langid)` and `$(vlang)`, which represent a directory or a filename in a path, only match inside or at the end of a provided path. For example: *english* is not replaced by *$(vlang)*, whereas *file:///c:/english* is replaced by *file:///c:/$(vlang)*.

The method `getSubstituteVariableValue()` returns the current value of the provided path variable as a predefined or a user-defined value. If an unknown variable name is provided, a `com.sun.star.container.NoSuchElementException` is thrown. The argument `variable` can be provided with preceding `"$("` and succeeding `")"` or without them. So both *$(work)* and *work* can be used.

This code example shows how to access, substitute, and resubstitute path variables by means of the OpenOffice.org API.

```
import com.sun.star.bridge.XUnoUrlResolver;
import com.sun.star.uno.UnoRuntime;
import com.sun.star.uno.XComponentContext;
import com.sun.star.lang.XMultiComponentFactory;
import com.sun.star.beans.XPropertySet;
import com.sun.star.beans.PropertyValue;

import com.sun.star.util.XStringSubstitution;
import com.sun.star.frame.TerminationVetoException;
import com.sun.star.frame.XTerminateListener;

/*
 * Provides example code how to access and use the
 * path substitution sercvice.
 */
public class PathSubstitutionTest extends java.lang.Object {

    /*
     * List of predefined path variables supported by
     * the path substitution service.
     */
    private static String[] predefinedPathVariables = {
        "$(home)","$(inst)","$(prog)","$(temp)","$(user)",
        "$(work)","$(path)","$(lang)","$(langid)","$(vlang)"
    };

    /*
     * @param args the command line arguments
     */
    public static void main(String[] args) {

        XComponentContext xRemoteContext = null;
        XMultiComponentFactory xRemoteServiceManager = null;
        XStringSubstitution xPathSubstService = null;

        try {
            // connect
            XComponentContext xLocalContext =
                com.sun.star.comp.helper.Bootstrap.createInitialComponentContext(null);
            XMultiComponentFactory xLocalServiceManager = xLocalContext.getServiceManager();
            Object urlResolver  = xLocalServiceManager.createInstanceWithContext(
                "com.sun.star.bridge.UnoUrlResolver", xLocalContext );
            XUnoUrlResolver xUnoUrlResolver = (XUnoUrlResolver) UnoRuntime.queryInterface(
                XUnoUrlResolver.class, urlResolver );
            Object initialObject = xUnoUrlResolver.resolve(
                "uno:socket,host=localhost,port=8100;urp;StarOffice.ServiceManager" );
```

```
            XPropertySet xPropertySet = (XPropertySet)UnoRuntime.queryInterface(
                XPropertySet.class, initialObject);
            Object context = xPropertySet.getPropertyValue("DefaultContext");
            xRemoteContext = (XComponentContext)UnoRuntime.queryInterface(
                XComponentContext.class, context);
            xRemoteServiceManager = xRemoteContext.getServiceManager();

            Object pathSubst = xRemoteServiceManager.createInstanceWithContext(
                "com.sun.star.comp.framework.PathSubstitution", xRemoteContext );
            xPathSubstService = (XStringSubstitution)UnoRuntime.queryInterface(
                XStringSubstitution.class, pathSubst);

            /* Work with path variables */
            workWithPathVariables( xPathSubstService );
        }
        catch (java.lang.Exception e){
            e.printStackTrace();
        }
        finally {
            System.exit(0);
        }
    }

    /*
     * Retrieve, resubstitute path variables
     * @param xPathSubstService the path substitution service
     */
    public static void workWithPathVariables( XStringSubstitution xPathSubstService )
    {
        if ( xPathSubstService != null ) {
            for ( int i=0; i<predefinedPathVariables.length; i++ ) {
                try {
                        /* Retrieve values for predefined path variables */
                        String aPath = xPathSubstService.getSubstituteVariableValue(
                                        predefinedPathVariables[i] );
                        System.out.println( "Variable="+ predefinedPathVariables[i] +
                                        " Path=" + aPath );

                        /* Check resubstitute */
                        String aValue = xPathSubstService.reSubstituteVariables( aPath );
                        System.out.println( "Path=" + aPath +
                                        " Variable=" + aValue );
                }
                catch ( com.sun.star.container.NoSuchElementException e) {
                    System.out.println( "NoSuchElementExceptio has been thrown accessing"
                                    + predefinedPathVariables[i]);
                }
            }
        }
    }
}
```

6.2.8 OpenOffice.org Single Sign-On API

Overview

Users of a client application that can communicate with a variety of services on a network may need to enter several passwords during a single session to access different services. This situation can be further exacerbated if the client application also requires the user to enter a password each time a particular network service is accessed during a session.

As most network users must authenticate to an OS at login time, it would make sense to access some of the required network services at this time as well. A solution to this problem is provided by the Single Sign-On (SSO) methodology, which is the ability to login in once and access several protected network services.

The best known SSO is the Kerberos network authentication protocol (see *rfc1510*). Kerberos functionality is commonly accessed through the Generic Security Service Application Program Interface (GSS-API, see *rfc2743*). Central to GSS-API is the concept of a security context, which is the "state of trust" that is initiated when a client (also known as *source* or *initiator*) identifies itself to a network service (also known as *target* or *acceptor*). If mutual authentication is supported, then the

service can also authenticate itself to the client. To establish a security context, security tokens are exchanged, processed, and verified between the client and the service. The client always initiates this exchange. Once established, a security context can be used to encrypt or decrypt subsequent client-service communications.

The OpenOffice.org SSO API is based on GSS-API. The SSO API supports the creation of security contexts on the client and the service side as well as the generation of the security tokens that are required for the exchange to complete the security context based authentication. The SSO API does not support the actual exchange of security tokens or the encryption or decryption of client-service communications in an established security context.

OpenOffice.org implements SSO in two different ways to authenticate with an LDAP server for configuration purposes. The first is Kerberos based and the second is a simple non-standard "cached username/password" SSO. The latter is provided as a fallback to support scenarios where no Kerberos server is available.

Implementing the OpenOffice.org SSO API

Implementing the OpenOffice.org SSO API involves creating security context instances (see `XSSOInitiatorContext` and `XSSOAcceptorContext` below) and using these instances to create and process security tokens. All of the OpenOffice.org SSO interfaces are available from the `::com::sun::star::auth` namespace. The major interfaces are shown in Illustration 50 and described below.

XSSOManagerFactory
: Represents the starting point for interaction with the SSO API. This interface is responsible for providing `XSSOManager` (described below) instances based on the user's configured security mechanism e.g. "KERBEROS".

XSSOManager
: This interface is responsible for the creation of unestablished security contexts for clients (`XSSOInitiatorContext`) and services (`XSSOAcceptorContext`). An `XSSOManager` instance "supports" a single security mechanism, that is, the context instances that are created by an `XSSOManager` instance only interact with a single security mechanism implementation.

XSSOInitiatorContext
: This interface represents a client-side security context that is unestablished when it is created. A single method, `init()`, is provided so that you can create an initial client-side security token that can be delivered to the relevant service and for processing or validating returned service-side security tokens (if mutual authentication is supported). The expected sequence of events for this client-side security context is:

  - The client calls `init()`, passes NULL as the parameter, receives an appropriate client-side security token in return.
  - The client sends the security token to the relevant service.
  - If the service successfully processes this token, the client is authenticated.
  - If mutual authentication is not supported, the client-side authentication sequence is now complete.
  - If mutual authentication is supported, the service sends a service-side security token to the client.
  - The client calls `init()` a second time and passes the returned service-side security token as a parameter. If the token is successfully passed, the service is authenticated.

XSSOAcceptorContext

This interface represents a service-side security context that is not established when it is created. A single method, `accept()`, is provided and is responsible for processing an initial client-side security token. If mutual authentication is supported, the method also generates a service-side security token for the client. The expected sequence of events for this service-side security context is:

- The service receives the client-side security token.
- The service calls `accept()`, passes the client-side security token as a parameter, and if successful, the client is authenticated.
- If mutual authentication is not supported, the service-side authentication sequence is now complete.
- If mutual authentication is supported, `accept()` returns a non-zero length service-side security token.
- The service sends the service-side security token to the client to authenticate the service.

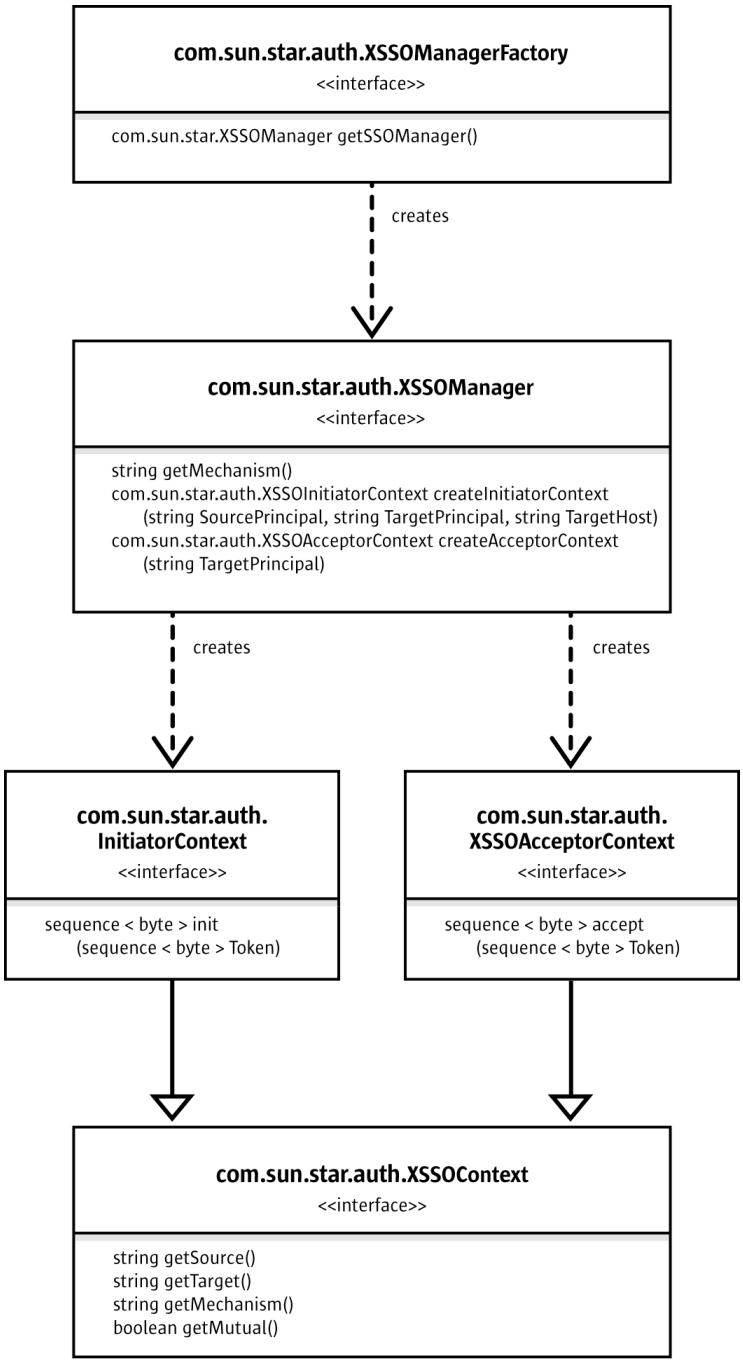


Illustration 71: Major Interfaces of the OpenOffice.org SSO

The following example is a sample usage of the OpenOffice.org SSO API that provides the `authenticate()` method of the fictitious client--side `MySSO` class. For simplicity, assume that `MySSO` has the following members:

- `mSourceName` identifies a client-side user that needs to authenticate to a network service.
- `mTargetName` identifies the service to which the user needs to authenticate.
- `mTargetHost` identifies the network host where the service of interest is running.

```
namespace auth   = ::com::sun::star::auth;
namespace lang   = ::com::sun::star::lang;
```

```
namespace uno    = ::com::sun::star::uno;

void MySSO::authenticate(void) {
    static const rtl::OUString kSSOService(
        RTL_CONSTASCII_USTRINGPARAM("com.sun.star.auth.SSOManagerFactory"));

    uno::Reference< lang::XMultiServiceFactory > theServiceFactory =
        ::comphelper::getProcessServiceFactory();

    // Create an SSO Manager Factory.
    uno::Reference< auth::XSSOManagerFactory > theSSOFactory(
        theServiceFactory->createInstance(kSSOService), uno::UNO_QUERY);
    if (!theSSOFactory.is()) {
        throw;
    }

    // Ask the SSO Manager Factory for an SSO Manager.
    uno::Reference<auth::XSSOManager> theSSOManager =
        theSSOFactory->getSSOManager();
   if (!theSSOManager.is()) {
        throw;
    }

    // Ask the SSO Manager to create an unestablished client/initiator side
    // security context based on user name, service name and service host.
    uno::Reference<auth::XSSOInitiatorContext> theInitiatorContext =
        theSSOManager->createInitiatorContext(mSourceName, mTargetName, mTargetHost);

    // Now create the client side security token to send to the service.
    uno::Sequence<sal_Int8> theClientToken = theInitiatorContext->init(NULL);

    // The client should now send 'theClientToken' to the service.
    // If mutual authentication is supported, the service will return a service
    // side security token.
    uno::Sequence<sal_Int8> theServerToken = sendToken(theClientToken);
    if (theInitiatorContext->getMutual()) {
        theInitiatorContext->init(theServerToken);
    }
}
```

The SSO Password Cache

When you implement the SSO API, you may require access to user passwords, especially if you are relying on a preexisting underlying security mechanism. If you do not know how to gain such access, you can use the OpenOffice.org SSO password cache. This cache provides basic support for maintaining a list of user name or password entries. Individual entries have a default lifetime corresponding to a single user session, but can optionally exist for multiple sessions. Support is provided for adding, retrieving, and deleting cache entries. Only one entry per user name can exist in the cache at any time. If you add an entry for an existing user name, the new entry replaces the original entry.

The SSO password cache is represented by a single interface, namely the `XSSOPasswordCache` interface, available in the `::com::sun::star::auth` namespace.

7 Text Documents

7.1 Overview

In the OpenOffice.org API, a text document is a document model which is able to handle text contents. A document in our context is a product of work that can be stored and printed to make the result of the work a permanent resource. By model we mean data that forms the basis of a document and is organized in a manner that allows working with the data independently from their visual representation in a graphical user interface.

It is important to understand that developers have to work with the model directly, when they want to change it through the OpenOffice.org API. The model *has* a controller object which enables developers to manipulate the visual presentation of the document in the user interface. But the controller is not used to change a document. The controller serves two purposes.

- The controller interacts with the user interface for movement, such as moving the visible text cursor, flipping through screen pages or changing the zoom factor.
- The second purpose is getting information about the current view status, such as the current selection, the current page, the total page count or the line count. Automatic page or line breaks are not really part of the document data, but rather something that is needed in a certain presentation of the document.

Keeping the difference between model and controller in mind, we will now discuss the parts of a text document model in the OpenOffice.org API.

The text document model in the OpenOffice.org API has five major architectural areas, cf. Illustration 72 below. The five areas are:

- text
- service manager (document internal)
- draw page
- text content suppliers
- objects for styling and numbering

The core of the text document model is the text. It consists of character strings organized in paragraphs and other text contents. The usage of text will be discussed in *7.3 Text Documents - Working with Text Documents*.

The service manager of the document model creates all text contents for the model, except for the paragraphs. Note that the document service manager is different from the main service manager that is used when connecting to the office. Each document model has its own service manager, so that the services can be adapted to the document when required. Examples for text contents

created by the text document service manager are text tables, text fields, drawing shapes, text frames or graphic objects. The service manager is asked for a text content, then you insert it into the text.

Afterwards, the majority of these text contents in a text can be retrieved from the model using text content suppliers. The exception are drawing shapes. They can be found on the `DrawPage`, which is discussed below.

Above the text lies the `DrawPage`. It is used for drawing contents. Imagine it as a transparent layer with contents that can affect the text under the layer, for instance by forcing it to wrap around contents on the `DrawPage`. However, text can also wrap through `DrawPage` contents, so the similarity is limited.

Finally, there are services that allow for document wide styling and structuring of the text. Among them are style family suppliers for paragraphs, characters, pages and numbering patterns, and suppliers for line and outline numbering.

Besides these five architectural areas, there are a number of aspects covering the document character of our model: It is printable, storable, modifiable, it can be refreshed, its contents are able to be searched and replaced and it supplies general information about itself. These aspects are shown at the lower right of the illustration.

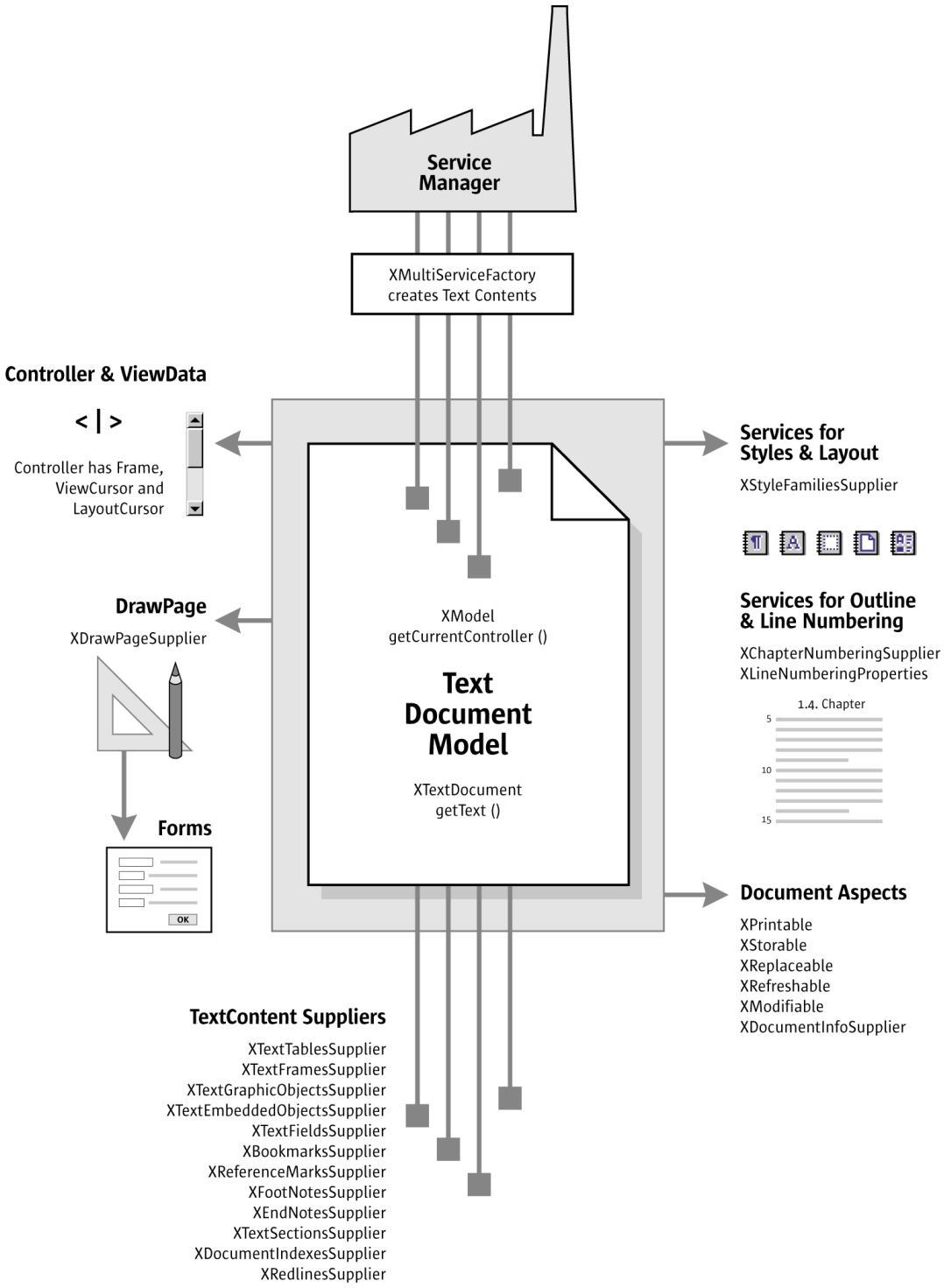


Illustration 72 Text Document Model

Finally, the controller provides access to the graphical user interface for the model and has knowledge about the current view status in the user interface, cf. the upper left of the diagram above.

The usage of text is discussed in the section *7.3.1 Text Documents - Working with Text Documents - Word Processing* below. This overview will be concluded by two examples:

7.1.1 Example: Fields in a Template

All following code samples are contained in *TextDocuments.java*. This file is located in the Samples folder that comes with the resources for the developer's manual.

The examples use the environment from chapter *2 First Steps*, for instance, connecting using the `getRemoteServiceManager()` method.

We want to use a template file containing text fields and bookmarks and insert text into the fields and at the cursor position. The suitable template file *TextTemplateWithUserFields.sxw* lies in the Samples folder, as well. Edit the path to this file below before running the sample.

The first step is to load the file as a template, so that OpenOffice.org creates a new, untitled document. As in the chapter *2 First Steps*, we have to connect, get the `Desktop` object, query its `XComponentLoader` interface and call `loadComponentFromUrl()`. This time we tell OpenOffice.org how it should load the file. The key for loading parameters is the sequence of `PropertyValue` structs passed to `loadComponentFromUrl()`. The appropriate `PropertyValue` name is `AsTemplate` and we have to set `AsTemplate` to `true`. (Text/TextDocuments.java)

```
/** Load a document as template */
protected XComponent newDocComponentFromTemplate(String loadUrl) throws java.lang.Exception {
    // get the remote service manager
    mxRemoteServiceManager = this.getRemoteServiceManager(unoUrl);
    // retrieve the Desktop object, we need its XComponentLoader
    Object desktop = mxRemoteServiceManager.createInstanceWithContext(
        "com.sun.star.frame.Desktop", mxRemoteContext);
    XComponentLoader xComponentLoader = (XComponentLoader)UnoRuntime.queryInterface(
        XComponentLoader.class, desktop);

    // define load properties according to com.sun.star.document.MediaDescriptor
    // the boolean property AsTemplate tells the office to create a new document
    // from the given file
    PropertyValue[] loadProps = new PropertyValue[1];
    loadProps[0] = new PropertyValue();
    loadProps[0].Name = "AsTemplate";
    loadProps[0].Value = new Boolean(true);
    // load
    return xComponentLoader.loadComponentFromURL(loadUrl, "_blank", 0, loadProps);
}
```

Now that we are able to load a text document as template, we will open an existing template file that contains five text fields and a bookmark. We want to demonstrate how to insert text at predefined positions in a document.

Text fields and bookmarks are supplied by the appropriate `XTextFieldsSupplier` and `XBookmarksSupplier` interfaces. Their fully qualified names are `com.sun.star.text.XTextFieldsSupplier` and `com.sun.star.text.XBookmarksSupplier`.

The `XTextFieldsSupplier` provides collections of text fields in our text. We use document variable fields for our purpose, which are`com.sun.star.text.textfield.User` services. All `User` fields have a field master that holds the actual content of the variable. Therefore, the `TextFields` collection, as well as the `FieldMasters` are required for our example. We get the field masters for the five fields by name and set their `Content` property. Finally, we refresh the text fields so that they reflect the changes made to the field masters.

The `XBookmarksSupplier` returns all bookmarks in our document. The collection of bookmarks is a `com.sun.star.container.XNameAccess`, so that the bookmarks are retrieved by name. Every object in a text supports the interface `XTextContent` that has a method `getAnchor()`. The anchor is the text range an object takes up, so `getAnchor()` retrieves is an `XTextRange`. From the chapter *2 First Steps*, a `com.sun.star.text.XTextRange` allows setting the string of a text range. Our bookmark is a text content and therefore must support `XTextContent`. Inserting text at a bookmark position is straightforward: get the anchor of the bookmark and set its string. (Text/TextDocuments.java)

```
/** Sample for use of templates
    This sample uses the file TextTemplateWithUserFields.sxw from the Samples folder.
```

```
    The file contains a number of User text fields (Variables - User) and a bookmark
    which we use to fill in various values
 */
protected void templateExample() throws java.lang.Exception {
    // create a small hashtable that simulates a rowset with columns
    Hashtable recipient = new Hashtable();
    recipient.put("Company", "Manatee Books");
    recipient.put("Contact", "Rod Martin");
    recipient.put("ZIP", "34567");
    recipient.put("City", "Fort Lauderdale");
    recipient.put("State", "Florida");

    // load template with User fields and bookmark
    XComponent xTemplateComponent = newDocComponentFromTemplate(
    "file:///X:/devmanual/Samples/TextTemplateWithUserFields.sxw");

    // get XTextFieldsSupplier and XBookmarksSupplier interfaces from document component
    XTextFieldsSupplier xTextFieldsSupplier = (XTextFieldsSupplier)UnoRuntime.queryInterface(
        XTextFieldsSupplier.class, xTemplateComponent);
    XBookmarksSupplier xBookmarksSupplier = (XBookmarksSupplier)UnoRuntime.queryInterface(
        XBookmarksSupplier.class, xTemplateComponent);

    // access the TextFields and the TextFieldMasters collections
    XNameAccess xNamedFieldMasters = xTextFieldsSupplier.getTextFieldMasters();
    XEnumerationAccess xEnumeratedFields = xTextFieldsSupplier.getTextFields();

    // iterate over hashtable and insert values into field masters
    java.util.Enumeration keys = recipient.keys();
    while (keys.hasMoreElements()) {
        // get column name
        String key = (String)keys.nextElement();

        // access corresponding field master
        Object fieldMaster = xNamedFieldMasters.getByName(
            "com.sun.star.text.FieldMaster.User." + key);

        // query the XPropertySet interface, we need to set the Content property
        XPropertySet xPropertySet = (XPropertySet)UnoRuntime.queryInterface(
            XPropertySet.class, fieldMaster);

        // insert the column value into field master
        xPropertySet.setPropertyValue("Content", recipient.get(key));
    }

    // afterwards we must refresh the textfields collection
    XRefreshable xRefreshable = (XRefreshable)UnoRuntime.queryInterface(
        XRefreshable.class, xEnumeratedFields);
    xRefreshable.refresh();

    // accessing the Bookmarks collection of the document
    XNameAccess xNamedBookmarks = xBookmarksSupplier.getBookmarks();

    // find the bookmark named "Subscription"
    Object bookmark = xNamedBookmarks.getByName("Subscription");

    // we need its XTextRange which is available from getAnchor(),
    // so query for XTextContent
    XTextContent xBookmarkContent = (XTextContent)UnoRuntime.queryInterface(
        XTextContent.class, bookmark);

    // get the anchor of the bookmark (its XTextRange)
    XTextRange xBookmarkRange = xBookmarkContent.getAnchor();

    // set string at the bookmark position
    xBookmarkRange.setString("subscription for the Manatee Journal");
}
```

7.1.2 Example: Visible Cursor Position

As discussed earlier, the OpenOffice.org API distinguishes between the model and controller. This difference is mirrored in two different kinds of cursors in the API: model cursors and visible cursors. The visible cursor is also called view cursor.

The second example assumes that the user has selected a text range in a paragraph and expects something to happen at that cursor position. Setting character and paragraph styles, and retrieving the current page number at the view cursor position is demonstrated in the example. The view cursor will be transformed into a model cursor.

We want to work with the current document, therefore we cannot use `loadComponentFromURL()`. Rather, we ask the com.sun.star.frame.Desktop service for the current component. Once we have the current component—which is our document model—we go from the model to the controller and get the view cursor.

The view cursor has properties for the current character and paragraph style. The example uses built-in styles and sets the property `CharStyleName` to `"Quotation"` and `ParaStyleName` to `"Quotations"`. Furthermore, the view cursor knows about the automatic page breaks. Because we are interested in the current page number, we get it from the view cursor and print it out.

The model cursor is much more powerful than the view cursor when it comes to possible movements and editing capabilities. We create a model cursor from the view cursor. Two steps are necessary: We ask the view cursor for its `Text` service, then we have the `Text` service create a model cursor based on the current cursor position. The model cursor knows where the paragraph ends, so we go there and insert a string. (Text/TextDocuments.java)

```
/** Sample for document changes, starting at the current view cursor position
    The sample changes the paragraph style and the character style at the current
    view cursor selection
    Open the sample file ViewCursorExampleFile, select some text and run the example
    The current paragraph will be set to Quotations paragraph style
    The selected text will be set to Quotation character style
 */
private void viewCursorExample() throws java.lang.Exception {
    // get the remote service manager
    mxRemoteServiceManager = this.getRemoteServiceManager(unoUrl);

    // get the Desktop service
    Object desktop = mxRemoteServiceManager.createInstanceWithContext(
        "com.sun.star.frame.Desktop", mxRemoteContext);

    // query its XDesktop interface, we need the current component
    XDesktop xDesktop = (XDesktop)UnoRuntime.queryInterface(
         XDesktop.class, desktop);

    // retrieve the current component and access the controller
    XComponent xCurrentComponent = xDesktop.getCurrentComponent();

    // get the XModel interface from the component
    XModel xModel = (XModel)UnoRuntime.queryInterface(XModel.class, xCurrentComponent);

    // the model knows its controller
    XController xController = xModel.getCurrentController();

    // the controller gives us the TextViewCursor
    // query the viewcursor supplier interface
    XTextViewCursorSupplier xViewCursorSupplier =
        (XTextViewCursorSupplier)UnoRuntime.queryInterface(
            XTextViewCursorSupplier.class, xController);

    // get the cursor
    XTextViewCursor xViewCursor = xViewCursorSupplier.getViewCursor();

    // query its XPropertySet interface, we want to set character and paragraph properties
    XPropertySet xCursorPropertySet = (XPropertySet)UnoRuntime.queryInterface(
        XPropertySet.class, xViewCursor);

    // set the appropriate properties for character and paragraph style
    xCursorPropertySet.setPropertyValue("CharStyleName", "Quotation");
    xCursorPropertySet.setPropertyValue("ParaStyleName", "Quotations");

    // print the current page number - we need the XPageCursor interface for this
    XPageCursor xPageCursor = (XPageCursor)UnoRuntime.queryInterface(
        XPageCursor.class, xViewCursor);
    System.out.println("The current page number is " + xPageCursor.getPage());

    // the model cursor is much more powerful, so
    // we create a model cursor at the current view cursor position with the following steps:
    // we get the Text service from the TextViewCursor, the cursor is an XTextRange and has
    // therefore a method getText()
    XText xDocumentText = xViewCursor.getText();

    // the text creates a model cursor from the viewcursor
    XTextCursor xModelCursor = xDocumentText.createTextCursorByRange(xViewCursor.getStart());

    // now we could query XWordCursor, XSentenceCursor and XParagraphCursor
    // or XDocumentInsertable, XSortable or XContentEnumerationAccess
    // and work with the properties of com.sun.star.text.TextCursor
```

```
    // in this case we just go to the end of the paragraph and add some text.
    XParagraphCursor xParagraphCursor = (XParagraphCursor)UnoRuntime.queryInterface(
    XParagraphCursor.class, xModelCursor);

    // goto the end of the paragraph
    xParagraphCursor.gotoEndOfParagraph(false);
    xParagraphCursor.setString(" ***** Fin de semana! ******");
}
```

7.2 Handling Text Document Files

7.2.1 Creating and Loading Text Documents

If a document in OpenOffice.org is required, begin by getting a `com.sun.star.frame.Desktop` service from the service manager. The desktop handles all document components in OpenOffice.org, among other things. It is discussed thoroughly in the chapter *6 Office Development*. Office documents are often called components, because they support the `com.sun.star.lang.XComponent` interface. An `XComponent` is a UNO object that can be disposed explicitly and broadcast an event to other UNO objects when this happens.

The `Desktop` can load new and existing components from a URL. For this purpose it has a `com.sun.star.frame.XComponentLoader` interface that has one single method to load and instantiate components from a URL into a frame:

```
com.sun.star.lang::XComponent loadComponentFromURL([in] string aURL,
        [in] string aTargetFrameName,
        [in] long nSearchFlags,
        [in] sequence< com::sun::star::beans::PropertyValue > aArgs );
```

The interesting parameters in our context are the URL that describes which resource should be loaded and the sequence of load arguments. For the target frame pass "`_blank`" and set the search flags to `0`. In most cases you will not want to reuse an existing frame.

The URL can be a `file:` URL, a `http:` URL, an `ftp:` URL or a `private:` URL. Look up the correct URL format in the load URL box in the function bar of OpenOffice.org. For new writer documents, a special URL scheme has to be used. The scheme is "private:", followed by "factory" as hostname. The resource is "swriter" for OpenOffice.org writer documents. For a new writer document, use "`private:factory/swriter`".

The load arguments are described in `com.sun.star.document.MediaDescriptor`. The arguments `AsTemplate` and `Hidden` have properties that are boolean values. If `AsTemplate` is `true`, the loader creates a new untitled document from the given URL. If it is `false`, template files are loaded for editing. If `Hidden` is `true`, the document is loaded in the background. This is useful when generating a document in the background without letting the user observe, for example, it can be used to generate a document and print it without previewing. *6 Office Development* describes other available options.

The section *7.1.1 Text Documents - Overview - Fields in a Template* discusses a complete example about how loading works. The following snippet loads a document in hidden mode: (Text/TextDocuments.java)

```
// (the method getRemoteServiceManager is described in the chapter First Steps)
mxRemoteServiceManager = this.getRemoteServiceManager(unoUrl);

// retrieve the Desktop object, we need its XComponentLoader
Object desktop = mxRemoteServiceManager.createInstanceWithContext(
"com.sun.star.frame.Desktop", mxRemoteContext);

// query the XComponentLoader interface from the Desktop service
XComponentLoader xComponentLoader = (XComponentLoader)UnoRuntime.queryInterface(
    XComponentLoader.class, desktop);
```

```
// define load properties according to com.sun.star.document.MediaDescriptor

/* or simply create an empty array of com.sun.star.beans.PropertyValue structs:
   PropertyValue[] loadProps = new PropertyValue[0]
*/

// the boolean property Hidden tells the office to open a file in hidden mode
PropertyValue[] loadProps = new PropertyValue[1];
loadProps[0] = new PropertyValue();
loadProps[0].Name = "Hidden";
loadProps[0].Value = new Boolean(true);

// load
return xComponentLoader.loadComponentFromURL(loadUrl, "_blank", 0, loadProps);
```

7.2.2 Saving Text Documents

Storing

Documents are storable through their interface `com.sun.star.frame.XStorable`. This interface is discussed in detail in *6 Office Development*. An `XStorable` implements these operations:

```
boolean hasLocation()
string getLocation()
boolean isReadonly()
void store()
void storeAsURL( [in] string aURL, sequence< com::sun::star::beans::PropertyValue > aArgs)
void storeToURL( [in] string aURL, sequence< com::sun::star::beans::PropertyValue > aArgs)
```

The method names are evident. The method `storeAsUrl()` is the exact representation of **File – Save As**, that is, it changes the current document location. In contrast, `storeToUrl()` stores a copy to a new location, but leaves the current document URL untouched.

Exporting

For exporting purposes, a filter name can be passed to `storeAsURL()` and `storeToURL()` that triggers an export to other file formats. The property needed for this purpose is the string argument `FilterName` that takes filter names defined in the configuration file:

<OfficePath>\share\config\registry\instance\org\openoffice\Office\TypeDetection.xml

In *TypeDetection.xml,* look for `<Filter/>` elements, their `cfg:name` attribute contains the needed strings for `FilterName`. The proper filter name for StarWriter 5.x is "StarWriter 5.0", and the export format "MS Word 97" is also popular. This is the element in *TypeDetection.xml* that describes the MS Word 97 filter:

```
<Filter cfg:name="MS Word 97">
   <Installed cfg:type="boolean">true</Installed>
   <UIName cfg:type="string" cfg:localized="true">
     <cfg:value xml:lang="en-US">Microsoft Word 97/2000/XP</cfg:value>
   </UIName>
   <Data cfg:type="string">3,writer_MS_Word_97,com.sun.star.text.TextDocument,,67,CWW8,0,,</Data>
 </Filter>
```

The following method stores a document using this filter: (Text/TextDocuments.java)

```
/** Store a document, using the MS Word 97/2000/XP Filter */
    protected void storeDocComponent(XComponent xDoc, String storeUrl) throws java.lang.Exception {

        XStorable xStorable = (XStorable)UnoRuntime.queryInterface(XStorable.class, xDoc);
        PropertyValue[] storeProps = new PropertyValue[1];
        storeProps[0] = new PropertyValue();
        storeProps[0].Name = "FilterName";
        storeProps[0].Value = "MS Word 97";
        xStorable.storeAsURL(storeUrl, storeProps);
    }
```

If an empty array of `PropertyValue` structs is passed, the native *.sxw* format of OpenOffice.org is used.

7.2.3 Printing Text Documents

Printer and Print Job Settings

Printing is a common office functionality. The chapter *6 Office Development* provides in-depth information about it. The writer document implements the `com.sun.star.view.XPrintable` interface for printing. It consists of three methods:

```
sequence< com::sun::star::beans::PropertyValue > getPrinter ()
void setPrinter ( [in] sequence< com::sun::star::beans::PropertyValue > aPrinter)
void print ( [in] sequence< com::sun::star::beans::PropertyValue > xOptions)
```

The following code is used with a given document `xDoc` to print to the standard printer without any settings: (Text/TextDocuments.java)

```
// query the XPrintable interface from your document
XPrintable xPrintable = (XPrintable)UnoRuntime.queryInterface(XPrintable.class, xDoc);

// create an empty printOptions array
PropertyValue[] printOpts = new PropertyValue[0];

// kick off printing
xPrintable.print(printOpts);
```

There are two groups of properties involved in general printing. The first one is used with `setPrinter()` and `getPrinter()` that controls the printer, and the second one is passed to `print()` and controls the print job.

`com.sun.star.view.PrinterDescriptor` comprises the properties for the printer:

Properties of `com.sun.star.view.PrinterDescriptor`	
`Name`	`string` — Specifies the name of the printer queue to be used.
`PaperOrientation`	`com.sun.star.view.PaperOrientation`. Specifies the orientation of the paper.
`PaperFormat`	`com.sun.star.view.PaperFormat`. Specifies a predefined paper size or if the paper size is a user-defined size.
`PaperSize`	`com.sun.star.awt.Size`. Specifies the size of the paper in 1/100 mm.
`IsBusy`	`boolean` — Indicates if the printer is busy.
`CanSetPaperOrientation`	`boolean` — Indicates if the printer allows changes to `PaperOrientation`.
`CanSetPaperFormat`	`boolean` — Indicates if the printer allows changes to PaperFormat.
`CanSetPaperSize`	`boolean` — Indicates if the printer allows changes to PaperSize.

`com.sun.star.view.PrintOptions` contains the following possibilities for a print job:

Properties of `com.sun.star.view.PrintOptions`	
`CopyCount`	`short` — Specifies the number of copies to print.
`FileName`	`string` — Specifies the name of a file to print to, if set.
`Collate`	`boolean` — Advises the printer to collate the pages of the copies. If `true`, a whole document is printed prior to the next copy, otherwise the page copies are completed together.

Properties of com.sun.star.view.PrintOptions	
Pages	string — Specifies the pages to print in the same format as in the print dialog of the GUI (e.g. "1, 3, 4-7, 9-")

The following method uses PrinterDescriptor and PrintOptions to print to a special printer, and preselect the pages to print. (Text/TextDocuments.java)

```
protected void printDocComponent(XComponent xDoc) throws java.lang.Exception {
    XPrintable xPrintable = (XPrintable)UnoRuntime.queryInterface(XPrintable.class, xDoc);
    PropertyValue[] printerDesc = new PropertyValue[1];
    printerDesc[0] = new PropertyValue();
    printerDesc[0].Name = "Name";
    printerDesc[0].Value = "5D PDF Creator";

    xPrintable.setPrinter(printerDesc);

    PropertyValue[] printOpts = new PropertyValue[1];
    printOpts[0] = new PropertyValue();
    printOpts[0].Name = "Pages";
    printOpts[0].Value = "3-5,7";

    xPrintable.print(printOpts);
}
```

Printing Multiple Pages on one Page

The interface com.sun.star.text.XPagePrintable is used to print more than one document page to a single printed page.

```
sequence< com::sun::star::beans::PropertyValue > getPagePrintSettings()
void setPagePrintSettings( [in] sequence< com::sun::star::beans::PropertyValue > aSettings)
void printPages( [in] sequence< com::sun::star::beans::PropertyValue > xOptions)
```

The first two methods getPagePrintSettings() and setPagePrintSettings() control the page printing. They use a sequence of com.sun.star.beans.PropertyValues whose possible values are defined in com.sun.star.text.PagePrintSettings:

Properties of com.sun.star.text.PagePrintSettings	
PageRows	short — Number of rows in which document pages should appear on the output page.
PageColumns	short — Number of columns in which document pages should appear on the output page.
LeftMargin	long — Left margin on the output page.
RightMargin	long — Right margin on the output page.
TopMargin	long — Top margin on the output page.
BottomMargin	long — Bottom margin on the output page.
HoriMargin	long — Margin between the columns on the output page.
VertMargin	long — Margin between the rows on the output page.
IsLandscape	boolean — Determines if the output page is in landscape format.

The method printPages() prints the document according to the previous settings. The argument for the printPages() method may contain the PrintOptions as described in the section above (containing the properties CopyCount, FileName, Collate and Pages).

7.3 Working with Text Documents

7.3.1 Word Processing

The text model in Illustration 72 shows that working with text starts with the method `getText()` at the `XTestDocument` interface of the document model. It returns a `com.sun.star.text.Text` service that handles text in OpenOffice.org.

The Text service has two mandatory interfaces and no properties:

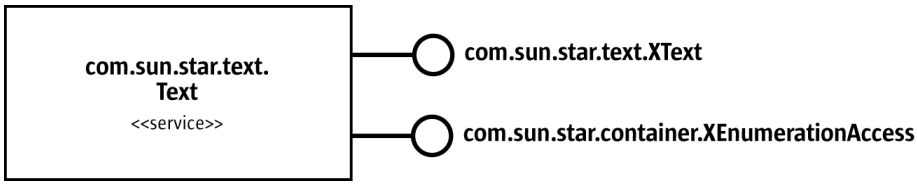


Illustration 73: Service com.sun.star.text.Text (mandatory interfaces only)

The `XText` is used to edit a text, and `XEnumerationAccess` is used to iterate over text. The following sections discuss these aspects of the Text service.

Editing Text

As previously discussed in the introductory chapter *2 First Steps*, the interface `com.sun.star.text.XText` incorporates three interfaces: `XText`, `XSimpleText` and `XTextRange`. When working with an `XText`, you work with the string it contains, or you insert and remove contents other than strings, such as tables, text fields, and graphics.

Strings

The `XText` is handled as a whole. There are two possibilities if the text is handled as one string. The complete string can be set at once, or strings can be added at the beginning or end of the existing text. These are the appropriate methods used for that purpose:

```
void setString( [in] string text)
String getString()
```

Consider the following example: (Text/TextDocuments.java)

```
/** Setting the whole text of a document as one string */
protected void BodyTextExample() {
    // Body Text and TextDocument example
    try {
        // demonstrate simple text insertion
        mxDocText.setString("This is the new body text of the document."
            + "\n\nThis is on the second line.\n\n");
    } catch (Exception e) {
        e.printStackTrace (System.out);
    }
}
```

Beginning and end of a text can be determined calling `getStart()` and `getEnd()`:

```
com::sun::star::text::XTextRange getStart()
com::sun::star::text::XTextRange getEnd()
```

The following example adds text using the start and end range of a text: (Text/TextDocuments.java)

```
/** Adding a string at the end or the beginning of text */
protected void TextRangeExample() {
```

```
    try {
        // Get a text range referring to the beginning of the text document
        XTextRange xStart = mxDocText.getStart();
        // use setString to insert text at the beginning
        xStart.setString ("This is text inserted at the beginning.\n\n");
        // Get a text range referring to the end of the text document
        XTextRange xEnd = mxDocText.getEnd();
        // use setString to insert text at the end
            xEnd.setString ("This is text inserted at the end.\n\n");
    } catch (Exception e) {
        e.printStackTrace(System.out);
    }
}
```

The above code is not very flexible. To gain flexibility, create a text cursor that is a movable text range. Note that such a text cursor is not visible in the user interface. The `XText` creates a cursor that works on the model immediately. The following methods can be used to get as many cursors as required:

```
com::sun::star::text::XTextCursor createTextCursor()
com::sun::star::text::XTextCursor createTextCursorByRange(
                          com::sun::star::text::XTextRange aTextPosition)
```

The text cursor travels through the text as a "collapsed" text range with identical start and end as a point in text, or it can expand while it moves to contain a target string. This is controlled with the methods of the `XTextCursor` interface:

```
// moving the cursor
// if bExpand is true, the cursor expands while it travels
boolean goLeft( [in] short nCount, [in] boolean bExpand)
boolean goRight( [in] short nCount, [in] boolean bExpand)
void gotoStart( [in] boolean bExpand)
void gotoEnd( [in] boolean bExpand)
void gotoRange( [in] com::sun::star::text::XTextRange xRange, [in] boolean bExpand)

// controlling the collapsed status of the cursor
void collapseToStart()
void collapseToEnd()
boolean isCollapsed()
```

In writer, a text cursor has three interfaces that inherit from `XTextCursor`: `com.sun.star.text.XWordCursor`, `com.sun.star.text.XSentenceCursor` and `com.sun.star.text.XParagraphCursor`. These interfaces introduce the following additional movements and status checks:

```
boolean gotoNextWord( [in] boolean bExpand)
boolean gotoPreviousWord( [in] boolean bExpand)
boolean gotoEndOfWord( [in] boolean bExpand)
boolean gotoStartOfWord( [in] boolean bExpand)
boolean isStartOfWord()
boolean isEndOfWord()

boolean gotoNextSentence( [in] boolean Expand)
boolean gotoPreviousSentence( [in] boolean Expand)
boolean gotoStartOfSentence( [in] boolean Expand)
boolean gotoEndOfSentence( [in] boolean Expand)
boolean isStartOfSentence()
boolean isEndOfSentence()

boolean gotoStartOfParagraph( [in] boolean bExpand)
boolean gotoEndOfParagraph( [in] boolean bExpand)
boolean gotoNextParagraph( [in] boolean bExpand)
boolean gotoPreviousParagraph( [in] boolean bExpand)
boolean isStartOfParagraph()
boolean isEndOfParagraph()
```

Since `XTextCursor` inherits from `XTextRange`, a cursor is an `XTextRange` and incorporates the methods of an `XTextRange`:

```
com::sun::star::text::XText getText()
com::sun::star::text::XTextRange getStart()
com::sun::star::text::XTextRange getEnd()
string getString()
void setString( [in] string aString)
```

The cursor can be told where it is required and the string content can be set later. This does have a drawback. After setting the string, the inserted string is always selected. That means further text can not be added without moving the cursor again. Therefore the most flexible method to insert strings by means of a cursor is the method `insertString()` in `XText`. It takes an `XTextRange` as the target range that is replaced during insertion, a string to insert, and a `boolean` parameter that determines if the inserted text should be absorbed by the cursor after it has been inserted. The `XTextRange` could be any `XTextRange`. The `XTextCursor` is an `XTextRange`, so it is used here:

```
void insertString( [in] com::sun::star::text::XTextRange xRange,
                   [in] string aString,
                   [in] boolean bAbsorb)
```

To insert text sequentially the `bAbsorb` parameter must be set to `false`, so that the `XTextRange` collapses at the end of the inserted string after insertion. If `bAbsorb` is `true`, the text range selects the new inserted string. The string that was selected by the text range prior to insertion is deleted.

Consider the use of `insertString()` below: (Text/TextDocuments.java)

```
/** moving a text cursor, selecting text and overwriting it */
protected void TextCursorExample() {
    try {
        // First, get the XSentenceCursor interface of our text cursor
        XSentenceCursor xSentenceCursor = (XSentenceCursor)UnoRuntime.queryInterface(
            XSentenceCursor.class, mxDocCursor);

        // Goto the next cursor, without selecting it
        xSentenceCursor.gotoNextSentence(false);

        // Get the XWordCursor interface of our text cursor
        XWordCursor xWordCursor = (XWordCursor) UnoRuntime.queryInterface(
            XWordCursor.class, mxDocCursor);

        // Skip the first four words of this sentence and select the fifth
        xWordCursor.gotoNextWord(false);
        xWordCursor.gotoNextWord(false);
        xWordCursor.gotoNextWord(false);
        xWordCursor.gotoNextWord(false);
        xWordCursor.gotoNextWord(true);

        // Use the XSimpleText interface to insert a word at the current cursor
        // location, over-writing
        // the current selection (the fifth word selected above)
        mxDocText.insertString(xWordCursor, "old ", true);

        // Access the property set of the cursor, and set the currently selected text
        // (which is the string we just inserted) to be bold
        XPropertySet xCursorProps = (XPropertySet) UnoRuntime.queryInterface(
            XPropertySet.class, mxDocCursor);
        xCursorProps.setPropertyValue("CharWeight", new Float(com.sun.star.awt.FontWeight.BOLD));

        // replace the '.' at the end of the sentence with a new string
        xSentenceCursor.gotoEndOfSentence(false);
        xWordCursor.gotoPreviousWord(true);
        mxDocText.insertString(xWordCursor,
            ", which has been changed with text cursors!", true);
    } catch (Exception e) {
            e.printStackTrace(System.out);
    }
}
```

Text Contents Other Than Strings

Up to this point, paragraphs made up of character strings has been discussed. Text can also contain other objects besides character strings in paragraphs. They all support the interface `com.sun.star.text.XTextContent`. In fact, everything in texts must support `XTextContent`.

A text content is an object that is attached to a `com.sun.star.text.XTextRange`. The text range it is attached to is called the *anchor* of the text content.

All text contents mentioned below, starting with tables, support the service `com.sun.star.text.TextContent`. It includes the interface `com.sun.star.text.XTextContent` that inherits from the interface `com.sun.star.lang.XComponent`. The `TextContent` services may have the following properties:

Properties of com.sun.star.text.TextContent	
AnchorType	Describes the base the object is positioned to, according to com.sun.star.text.TextContentAnchorType.
AnchorTypes	A sequence of com.sun.star.text.TextContentAnchorType that contains all allowed anchor types for the object.
TextWrap	Determines the way the surrounding text flows around the object, according to com.sun.star.text.WrapTextMode.

The method dispose() of the XComponent interface deletes the object from the document. Since a text content is an XComponent, com.sun.star.lang.XEventListener can be added or removed with the methods addEventListener() and removeEventListener(). These methods are called back when the object is disposed. Other events are not supported.

The method getAnchor() at the XTextContent interface returns a text range which reflects the text position where the object is located. This method may return a void object, for example, for text frames that are bound to a page. The method getAnchor() is used in situations where an XTextRange is required. For instance, placeholder fields (com.sun.star.text.textfield.JumpEdit) can be filled out using their getAnchor() method. Also, yo can get a bookmark, retrieve its XTextRange from getAnchor() and use it to insert a string at the bookmark position.

The method attach() is an intended method to attach text contents to the document, but it is currently not implemented.

All text contents—*including* paragraphs—can be created by the service manager of the document. They are created using the factory methods createInstance() or createInstanceWithArguments() at the com.sun.star.lang.XMultiServiceFactory interface of the document.

All text contents—*except* for paragraphs—can be inserted into text using the com.sun.star.text.XText method insertTextContent(). They can be removed by calling removeTextContent(). Starting with the section *7.3.4 Text Documents - Working with Text Documents - Tables*, there are code samples showing the usage of the document service manager with insertTextContent().

```
void insertTextContent( [in] com::sun::star::text::XTextRange xRange,
                        [in] com::sun::star::text::XTextContent xContent, boolean bAbsorb);
void removeTextContent( [in] com::sun::star::text::XTextContent xContent)
```

Paragraphs cannot be inserted by insertTextContent(). Only the interface XRelativeTextContentInsert can insert paragraphs. A paragraph created by the service manager can be used for creating a new paragraph before or after a table, or a text section positioned at the beginning or the end of page where no cursor can insert new paragraphs. Cf. the section *7.3.1 Text Documents - Working with Text Documents - Word Processing - Inserting a Paraqraph where no Cursor can go*below.

Control Characters

We have used Java escape sequences for paragraph breaks, but this may not be feasible in every language. Moreover, OpenOffice.org supports a number of control characters that can be used. There are two possibilities: use the method

```
void insertControlCharacter( [in] com::sun::star::text::XTextRange xRange,
                             [in] short nControlCharacter,
                             [in] boolean bAbsorb)
```

to insert single control characters as defined in the constants group com.sun.star.text.ControlCharacter, or use the corresponding unicode character from the following list as escape sequence in a string if your language supports it. In Java, Unicode characters in strings can be incorporated using the \uHHHH escape sequence, where H represents a hexadecimal digit

PARAGRAPH_BREAK	Insert a paragraph break (UNICODE 0x000D).
LINE_BREAK	Inserts a line break inside of the paragraph (UNICODE 0x000A).
HARD_HYPHEN	A character that appears like a dash, but prevents hyphenation at its position (UNICODE 0x2011).
SOFT_HYPHEN	Marks a preferred position for hyphenation (UNICODE 0x00AD).
HARD_SPACE	A character that appears like a space, but prevents hyphenation at this point (UNICODE 0x00A0).
APPEND_PARAGRAPH	A new paragraph is appended (no UNICODE for this function).

The section *7.3.2 Text Documents - Working with Text Documents - Formatting* describes how page breaks are created by setting certain paragraph properties.

Iterating over Text

The second interface of `com.sun.star.text.Text` is `XEnumerationAccess`. AText service enumerates all paragraphs in a text and returns objects which support `com.sun.star.text.Paragraph`. This includes tables, because writer sees tables as specialized paragraphs that support the `com.sun.star.text.TextTable` service.

Paragraphs also have an `com.sun.star.container.XEnumerationAccess` of their own. They can enumerate every single text portion that they contain. A text portion is a text range containing a uniform piece of information that appears within the text flow. An ordinary paragraph, formatted in a uniform manner and containing nothing but a string, enumerates just a single text portion. In a paragraph that has specially formatted words or other contents, the text portion enumeration returns one `com.sun.star.text.TextPortion` service for each differently formatted string, and for every other text content. Text portions include the service `com.sun.star.text.TextRange` and have the properties listed below:

Properties of `com.sun.star.text.TextPortion`	
`TextPortionType`	`string` — Contains the type of the text portion (see below).
`ControlCharacter`	`short` — Returns the control character if the text portion contains a control character as defined in `com.sun.star.text.ControlCharacter`.
`Bookmark`	`com.sun.star.text.XTextContent`. Contains the bookmark if the portion has TextPortionType="Bookmark".
`IsCollapsed`	`boolean` — Determines whether the portion is a point only.
`IsStart`	`boolean` — Determines whether the portion is a start portion if two portions are needed to include an object, that is, `DocmentIndexMark`.

Possible Values for `TextPortionType` are:

TextPortionType (String)	**Description**
“Text”	a portion with mere string content
“TextField”	A `com.sun.star.text.TextField` content.
“TextContent”	A text content supplied through the interface `XContentEnumerationAccess`.
“Footnote”	A footnote or an endnote.
“ControlCharacter”	A control character.
“ReferenceMark”	A reference mark.

TextPortionType (String)	Description
“DocumentIndexMark”	A document index mark.
“Bookmark”	A bookmark.
“Redline”	A redline portion which is a result of the change tracking feature.
“Ruby”	A ruby attribute which is used in Asian text.

The text portion enumeration of a paragraph does not supply contents which do belong to the paragraph, but do not fuse together with the text flow. These could be text frames, graphic objects, embedded objects or drawing shapes anchored at the paragraph, characters or as character. The `TextPortionType` `"TextContent"` indicate if there is a content anchored at a character or as a character. If you have a `TextContent` portion type, you know that there are shape objects anchored at a character or as a character.

This last group of data contained in a text, `Paragraphs` and `TextPortions` in writer support the interface `com.sun.star.container.XContentEnumerationAccess`. This interface tells which text contents besides the text flow contents there are and supplies them as an `com.sun.star.container.XEnumeration`:

```
sequence< string > getAvailableServiceNames()
com::sun::star::container::XEnumeration createContentEnumeration( [in] string aServiceName)
```

The `XContentEnumerationAccess` of the paragraph lists the shape objects anchored at the paragraph while the `XContentEnumerationAccess` lists the shape objects anchored at a character or as a character.



Precisely the same enumerations are available for the current text cursor selection. The text cursor enumerates paragraphs, text portions and text contents just like the service `com.sun.star.text.Text` itself.

The enumeration access to text through paragraphs and text portions is used if every single paragraph in a text needs to be touched. The application area for this enumeration are export filters, that uses this enumeration to go over the whole document, writing out the paragraphs to the target file. The following code snippet centers all paragraphs in a text. (Text/TextDocuments.java)

```
/** This method demonstrates how to iterate over paragraphs */
protected void ParagraphExample () {
    try {
        // The service 'com.sun.star.text.Text' supports the XEnumerationAccess interface to
        // provide an enumeration
        // of the paragraphs contained by the text the service refers to.

        // Here, we access this interface
        XEnumerationAccess xParaAccess = (XEnumerationAccess) UnoRuntime.queryInterface(
            XEnumerationAccess.class, mxDocText);
        // Call the XEnumerationAccess's only method to access the actual Enumeration
        XEnumeration xParaEnum = xParaAccess.createEnumeration();

        // While there are paragraphs, do things to them
        while (xParaEnum.hasMoreElements()) {
            // Get a reference to the next paragraphs XServiceInfo interface. TextTables
            // are also part of this
            // enumeration access, so we ask the element if it is a TextTable, if it
            // doesn't support the
            // com.sun.star.text.TextTable service, then it is safe to assume that it
            // really is a paragraph
            XServiceInfo xInfo = (XServiceInfo) UnoRuntime.queryInterface(
                XServiceInfo.class, xParaEnum.nextElement());
            if (!xInfo.supportsService("com.sun.star.text.TextTable")) {
                // Access the paragraph's property set...the properties in this
                // property set are listed
                // in: com.sun.star.style.ParagraphProperties
                XPropertySet xSet = (XPropertySet) UnoRuntime.queryInterface(
                    XPropertySet.class, xInfo);
                // Set the justification to be center justified
                xSet.setPropertyValue("ParaAdjust", com.sun.star.style.ParagraphAdjust.CENTER);
            }
        }
    } catch (Exception e) {
```

```
                e.printStackTrace (System.out);
    }
}
```

Inserting a Paragraph where no Cursor can go

The service `com.sun.star.text.Text` has an optional interface `com.sun.star.text.XRelativeTextContentInsert` which is available in Text services in writer. The intention of this interface is to insert paragraphs in positions where no cursor or text portion can be located to use the `insertTextContent()` method. These situation occurs when text sections or text tables are at the start or end of the document, or if they follow each other directly.

```
void insertTextContentBefore( [in] com::sun::star::text::XTextContent xNewContent,
                              [in] com::sun::star::text::XTextContent xSuccessor)
void insertTextContentAfter( [in] com::sun::star::text::XTextContent xNewContent,
                             [in] com::sun::star::text::XTextContent xPredecessor)
```

The only supported text contents are `com.sun.star.text.Paragraph` as new content, and `com.sun.star.text.TextSection` and `com.sun.star.text.TextTable` as successor or predecessor.

Sorting Text

It is possible to sort text or the content of text tables.

Sorting of text is done by the text cursor that supports `com.sun.star.util.XSortable`. It contains two methods:

```
sequence< com::sun::star::beans::PropertyValue > createSortDescriptor()
void sort( [in] sequence< com::sun::star::beans::PropertyValue > xDescriptor)
```

The method `createSortDescriptor()` returns a sequence of `com.sun.star.beans.PropertyValue` that provides the elements as described in the service `com.sun.star.text.TextSortDescriptor`

The method `sort()` sorts the text that is selected by the cursor, by the given parameters.

Sorting of tables happens directly at the table service, which supports `XSortable`. Sorting is a common feature of OpenOffice.org and it is described in detail in *6 Office Development*.

Inserting Text Files

The text cursor in writer supports the interface `com.sun.star.document.XDocumentInsertable` which has a single method to insert a file at the current cursor position:

```
void insertDocumentFromURL( [in] string aURL,
                            [in] sequence< com::sun::star::beans::PropertyValue > aOptions)
```

Pass a URL and an empty sequence of `PropertyValue` structs. However, load properties could be used as described in `com.sun.star.document.MediaDescriptor`.

Auto Text

The auto text function can be used to organize reusable text passages. They allow storing text, including the formatting and all other contents in a text block collection to apply them later. Three services deal with auto text in OpenOffice.org:

- `com.sun.star.text.AutoTextContainer` specifies the entire collection of auto texts

- `com.sun.star.text.AutoTextGroup` describes a category of auto texts
- `com.sun.star.text.AutoTextEntry` is a single auto text. (Text/TextDocuments.java)

```
/** Insert an autotext at the current cursor position of given cursor mxDocCursor*/

// Get an XNameAccess interface to all auto text groups from the document factory
XNameAccess xContainer = (XNameAccess) UnoRuntime.queryInterface(
    XNameAccess.class, mxFactory.createInstance("com.sun.star.text.AutoTextContainer"));

// Get the autotext group Standard
xGroup = (XAutoTextGroup) UnoRuntime.queryInterface(
    XAutoTextGroup.class, xContainer.getByName("Standard"));

// get the entry Best Wishes (BW)
XAutoTextEntry xEntry = (XAutoTextEntry)UnoRuntime.queryInterface (
    XAutoTextEntry.class, xGroup.getByName ("BW"));

// insert the modified autotext block at the cursor position
xEntry.applyTo(mxDocCursor);



/** Add a new autotext entry to the AutoTextContainer
*/
// Select the last paragraph in the document
xParaCursor.gotoPreviousParagraph(true);

// Get the XAutoTextContainer interface of the AutoTextContainer service
XAutoTextContainer xAutoTextCont = (XAutoTextContainer) UnoRuntime.queryInterface(
    XAutoTextContainer.class, xContainer );

// If the APIExampleGroup already exists, remove it so we can add a new one
if (xContainer.hasByName("APIExampleGroup"))
    xAutoTextCont.removeByName("APIExampleGroup" );

// Create a new auto-text group called APIExampleGroup
XAutoTextGroup xNewGroup = xAutoTextCont.insertNewByName ( "APIExampleGroup" );

// Create and insert a new auto text entry containing the current cursor selection
XAutoTextEntry xNewEntry = xNewGroup.insertNewByName(
    "NAE", "New AutoTextEntry", xParaCursor);

// Get the XSimpleText and XText interfaces of the new autotext block
    XSimpleText xSimpleText = (XSimpleText) UnoRuntime.queryInterface(
        XSimpleText.class, xNewEntry);
XText xText = (XText) UnoRuntime.queryInterface(XText.class, xNewEntry);

// Insert a string at the beginning of the autotext block
xSimpleText.insertString(xText.getStart(),
    "This string was inserted using the API!\n\n", false);
```

The current implementation forces the user to close the `AutoTextEntry` instance when they are changed, so that the changes can take effect. However, the new AutoText is not written to disk until the destructor of the `AutoTextEntry` instance inside the writer is called. When this example has finished executing, the file on disk correctly contains the complete text `"This string was inserted using the API!\n\nSome text for a new autotext block"`, but there is no way in Java to call the destructor. It is not clear when the garbage collector deletes the object and writes the modifications to disk.

7.3.2 Formatting

A multitude of character, paragraph and other properties are available for text in OpenOffice.org. However, the objects implemented in the writer do not provide properties that support `com.sun.star.beans.XPropertyChangeListener` or `com.sun.star.beans.XVetoableChangeListener` yet.

Character and paragraph properties are available in the following services:

Services supporting Character and Paragraph Properties	Remark
com.sun.star.text.TextCursor	If collapsed, the CharacterProperties refer to the position on the right hand side of the cursor.
com.sun.star.text.Paragraph	
com.sun.star.text.TextPortion	
com.sun.star.text.TextTableCursor	
com.sun.star.text.Shape	
com.sun.star.table.CellRange	In text tables.
com.sun.star.text.TextDocument	The model offers a selected number of character properties which apply to the entire document. These are: CharFontName,CharFontStyleName, CharFontFamily, CharFontCharSet, CharFontPitch and their Asian counterparts CharFontStyleNameAsian, CharFontFamilyAsian, CharFontCharSetAsian, CharFontPitchAsian.

The character properties are described in the services
com.sun.star.style.CharacterProperties,
com.sun.star.style.CharacterPropertiesAsian and
com.sun.star.style.CharacterPropertiesComplex.

com.sun.star.style.CharacterProperties describes common character properties for all language zones and character properties in Western text. The following table provides possible values.

Properties of com.sun.star.style.CharacterProperties	
CharFontName	string — This property specifies the name of the font in western text.
CharFontStyleName	string — This property contains the name of the font style.
CharFontFamily	short — This property contains font family that is specified in com.sun.star.awt.FontFamily. Possible values are: DONTKNOW, DECORATIVE, MODERN, ROMAN, SCRIPT, SWISS, and SYSTEM.
CharFontCharSet	short — This property contains the text encoding of the font that is specified in com.sun.star.awt.CharSet. Possible values are: DONTKNOW, ANSI MAC, IBMPC_437, IBMPC_850, IBMPC_860, IBMPC_861, IBMPC_863, IBMPC_865, and SYSTEM SYMBOL.
CharFontPitch	short — This property contains the font pitch that is specified in com.sun.star.awt.FontPitch. The word font pitch refers to characters per inch, but the possible values are DONTKNOW, FIXED and VARIABLE. VARIABLE points to the difference between proportional and unproportional fonts.
CharColor	long — This property contains the value of the text color in ARGB notation. ARGB has four bytes denoting alpha, red, green and blue. In hex notation, this can be used conveniently: 0xAARRGGBB. The AA (Alpha) can be 00 or left out.
CharEscapement	[optional] short — Property which contains the relative value of the character height in subscription or superscription.
CharHeight	float — This value contains the height of the characters in point.

Properties of `com.sun.star.style.CharacterProperties`	
`CharUnderline`	`short` — This property contains the value for the character underline that is specified in `com.sun.star.awt.FontUnderline`. A lot of underline types are available. Some possible values are SINGLE, DOUBLE, and DOTTED.
`CharWeight`	`float` — This property contains the value of the font weight, cf. [`com.sun.star.awt.FontWeight`. A lot of weights are possible. The common ones are BOLD and NORMAL.
`CharPosture`	`long` — This property contains the posture of the font as defined in `com.sun.star.awt.FontSlant`. The most common values are ITALIC and NONE.
`CharAutoKerning`	`[optional] boolean` — Property to determine whether the kerning tables from the current font are used.
`CharBackColor`	`[optional] long` —Property which contains the text background color in ARGB: `0xAARRGGBB`.
`CharBackTransparent`	`[optional] boolean` —Determines if the text background color is set to transparent.
`CharCaseMap`	`[optional] short` — Property which contains the value of the case-mapping of the text for formatting and displaying. Possible CaseMaps are NONE, UPPERCASE, LOWERCASE, TITLE, and SMALLCAPS as defined in the constants group `com.sun.star.style.CaseMap`. (optional)
`CharCrossedOut`	`[optional] boolean` — This property is `true` if the characters are crossed out.
`CharFlash`	`[optional] boolean` — If this optional property is `true` , then the characters are flashing
`CharStrikeout`	`[optional] short` — Determines the type of the strikethrough of the character as defined in `com.sun.star.awt.FontStrikeout`. Values are NONE, SINGLE, DOUBLE, DONTKNOW, BOLD, and SLASH X.
`CharWordMode`	`[optional] boolean` — If this property is `true` , the underline and strike-through properties are not applied to white spaces.
`CharKerning`	`[optional] short` — Property which contains the value of the kerning of the characters.
`CharLocale`	struct `com.sun.star.lang.Locale`. Contains the locale (language and country) of the characters.
`CharKeepTogether`	`[optional] boolean` — Property which marks a range of characters to prevent it from being broken into two lines.
`CharNoLineBreak`	`[optional] boolean` — Property which marks a range of characters to ignore a line break in this area.
`CharShadowed`	`[optional] boolean` — `True` if the characters are formatted and displayed with a shadow effect. (optional)
`CharFontType`	`[optional] short` — Property which specifies the fundamental technology of the font as specified in `com.sun.star.awt.FontType`. Possible values are DONTKNOW, RASTER, DEVICE, and SCALABLE.
`CharStyleName`	`[optional] string` — Specifies the name of the style of the font.
`CharContoured`	`[optional] boolean` — `True` if the characters are formatted and displayed with a contour effect.
`CharCombineIsOn`	`[optional] boolean` — `True` if text is formatted in two lines.

Properties of com.sun.star.style.CharacterProperties	
CharCombinePrefix	[optional] string — Contains the prefix string (usually parenthesis) before text that is formatted in two lines.
CharCombineSuffix	[optional] string — Contains the suffix string (usually parenthesis) after text that is formatted in two lines.
CharEmphasize	[optional] short — Contains the font emphasis value com.sun.star.text.FontEmphasis.
CharRelief	[optional] short — Contains the relief value as FontRelief.
RubyText	[optional] string — Contains the text that is set as ruby.
RubyAdjust	[optional] short — Determines the adjustment of the ruby text as RubyAdjust.
RubyCharStyleName	[optional] string — Contains the name of the character style that is applied to RubyText (optional).
RubyIsAbove	[optional] boolean — Determines whether the ruby text is printed above/left or below/right of the text (optional) .
CharRotation	[optional] short — Determines the rotation of a character in degree.
CharRotationIsFitTo-Line	[optional] short — Determines whether the text formatting tries to fit rotated text into the surrounded line height.
CharScaleWidth	[optional] short — Determines the percentage value of scaling of characters.

com.sun.star.style.CharacterPropertiesAsian describes properties used in Asian text. All of these properties have a counterpart in CharacterProperties. They apply as soon as a text is recognized as Asian by the employed Unicode character subset.

Properties of com.sun.star.style.CharacterPropertiesAsian	
CharHeightAsian	float — This value contains the height of the characters in point.
CharWeightAsian	float — This property contains the value of the font weight.
CharFontNameAsian	string — This property specifies the name of the font style.
CharFontStyleNameAsian	string — This property contains the name of the font style.
CharFontFamilyAsian	short — This property contains the font family that is specified in com.sun.star.awt.FontFamily.
CharFontCharSetAsian	short — This property contains the text encoding of the font that is specified in com.sun.star.awt.CharSet.
CharFontPitchAsian	short — This property contains the font pitch that is specified in com.sun.star.awt.FontPitch.
CharPostureAsian	long — This property contains the value of the posture of the font as defined in com.sun.star.awt.FontSlant.
CharLocaleAsian	struct com.sun.star.lang.Locale — Contains the value of the locale.

The complex properties com.sun.star.style.CharacterPropertiesComplex refer to the same character settings as in CharacterPropertiesAsian, only they have the suffix “Complex” instead of “Asian”.

com.sun.star.style.ParagraphProperties comprises paragraph properties.

Properties of `com.sun.star.style.ParagraphProperties`	
`ParaAdjust`	`long` — Determines the adjustment of a paragraph.
`ParaLineSpacing`	`[optional]` struct `com.sun.star.style.LineSpacing` — Determines the line spacing of a paragraph.
`ParaBackColor`	`[optional] long` — Contains the paragraph background color.
`ParaBackTransparent`	`[optional] boolean` — This value is `true` if the paragraph background color is set to transparent.
`ParaBackGraphicURL`	`[optional] string` — Contains the value of a link for the background graphic of a paragraph.
`ParaBackGraphicFilter`	`[optional] string` — Contains the name of the graphic filter for the background graphic of a paragraph.
`ParaBackGraphicLocation`	`[optional] long` — Contains the value for the position of a background graphic according to `com.sun.star.style.GraphicLocation`.
`ParaLastLineAdjust`	`short` — Determines the adjustment of the last line.
`ParaExpandSingleWord`	`[optional] boolean` — Determines if single words are stretched.
`ParaLeftMargin`	`long` — Determines the left margin of the paragraph in 1/100 mm.
`ParaRightMargin`	`long` — Determines the right margin of the paragraph in 1/100 mm.
`ParaTopMargin`	`long` — Determines the top margin of the paragraph in 1/100 mm.
`ParaBottomMargin`	`long` — Determines the bottom margin of the paragraph in 1/100 mm.
`ParaLineNumberCount`	`[optional] boolean` — Determines if the paragraph is included in the line numbering.
`ParaLineNumberStartValue`	`[optional] boolean` — Contains the start value for the line numbering.
`ParaIsHyphenation`	`[optional] boolean` — Prevents the paragraph from getting hyphenated.
`PageDescName`	`[optional] string` — If this property is set, it creates a page break before the paragraph it belongs to and assigns the value as the name of the new page style sheet to use.
`PageNumberOffset`	`[optional] short` — If a page break property is set at a paragraph, this property contains the new value for the page number.
`ParaRegisterModeActive`	`[optional] boolean` — Determines if the register mode is applied to a paragraph.
`ParaTabStops`	`[optional] sequence < com.sun.star.style.TabStop >`. Specifies the positions and kinds of the tab stops within this paragraph.
`ParaStyleName`	`[optional] string` — Contains the name of the current paragraph style.
`DropCapFormat`	`[optional]` struct `com.sun.star.style.DropCapFormat` specifies whether the first characters of the paragraph are displayed in capital letters and how they are formatted.
`DropCapWholeWord`	`[optional] boolean` — Specifies if the property `DropCapFormat` is applied to the whole first word.
`ParaKeepTogether`	`[optional] boolean` — Setting this property to `true` prevents page or column breaks between this and the following paragraph.
`ParaSplit`	`[optional] boolean` — Setting this property to `false` prevents the paragraph from getting split into two pages or columns.
`NumberingLevel`	`[optional] short` — Specifies the numbering level of the paragraph.

Properties of com.sun.star.style.ParagraphProperties	
NumberingRules	com.sun.star.container.XIndexReplace. Contains the numbering rules applied to this paragraph.
NumberingStartValue	[optional] short — Specifies the start value for numbering if a new numbering starts at this paragraph.
ParaIsNumberingRestart	[optional] boolean — Determines if the numbering rules restart, counting at the current paragraph.
NumberingStyleName	[optional] string — Specifies the name of the style for the numbering.
ParaOrphans	[optional] byte — Specifies the minimum number of lines of the paragraph that have to be at bottom of a page if the paragraph is spread over more than one page.
ParaWidows	[optional] byte — Specifies the minimum number of lines of the paragraph that have to be at top of a page if the paragraph is spread over more than one page.
ParaShadowFormat	[optional] struct com.sun.star.table.ShadowFormat. Determines the type, color, and size of the shadow.
IsHangingPunctuation	[optional] boolean — Determines if hanging punctuation is allowed.
IsCharacterDistance	[optional] boolean — Determines if a distance between Asian text, western text or complex text is set.
IsForbiddenRules	[optional] boolean — Determines if the the rules for forbidden characters at the start or end of text lines are considered.

Objects supporting these properties support com.sun.star.beans.XPropertySet, as well. To change the properties, use the method setPropertyValue().

```
/** This snippet shows the necessary steps to set a property at the
    current position of a given text cursor mxDocCursor
*/

// query the XPropertySet interface
XPropertySet xCursorProps = (XPropertySet) UnoRuntime.queryInterface(XPropertySet.class, mxDocCursor);

// call setPropertyValue, passing in a Float object
xCursorProps.setPropertyValue("CharWeight", new Float ( com.sun.star.awt.FontWeight.BOLD));
```

The same procedure is used for all properties. The more complex properties are described here.

If a change of the page style is required the paragraph property PageDescName has to be set using an existing page style name. This forces a page break at the cursor position and the new inserted page uses the requested page style. The property PageNumberOffset has to be set to start with a new page count.

If a page break (or a column break) without a change in the used style is required, the property BreakType is set using the values of com.sun.star.style.BreakType:

Page break	Description
BreakType	Page or column break as described in com.sun.star.style.BreakType. Possible values are NONE, COLUMN_BEFORE, COLUMN_AFTER, COLUMN_BOTH, PAGE_BEFORE, PAGE_AFTER, and PAGE_BOTH. Setting the property forces a page or column break at the current text cursor position, paragraph or text table.

The property ParaLineNumberCount is used to include a paragraph in the line numbering. The setting of the line numbering options is done using the property set provided by the com.sun.star.text.XLineNumberingProperties interface implemented at the text document model.

To create a hyperlink these properties are set at the current cursor position or the current `com.sun.star.text.Paragraph` service.

Hyperlink properties are not specified for paragraphs in the API reference.

Hyperlink Properties	Description
`HyperLinkURL`	`string` — Contains the URL.
`HyperLinkTarget`	`string` — Contains the name of the target frame and can be left blank.
`HyperLinkName`	`string` — The name of the hyperlink can be left blank.
`UnvisitedCharStyleName`	`string` — The names of the character styles used to emphasize visited or not visited links. If left blank, the default character styles Internet Link/Visited Internet Link are applied automatically.
`VisitedCharStyleName`	
`HyperLinkEvents`	Events attached to the hyperlink. The names of the events are OnClick, OnMouseOver, and OnMouseOut. Each returned event is a sequence of `com.sun.star.beans.PropertyValue`, with three elements named EventType, MacroName and Library. All elements contain string values. The EventType contains the value "StarBasic" for OpenOffice.org Basic macros . The macro name contains the path to the macro, for example, Standard.Module1.Main. The library contains the name of the library.

Some properties are connected with each other. There may be side effects or dependencies between the following properties:

Interdependencies between Properties
`ParaRightMargin, ParaLeftMargin, ParaFirstLineIndent, ParaIsAutoFirstLineIndent`
`ParaTopMargin, ParaBottomMargin`
`ParaGraphicURL/Filter/Location, ParaBackColor, ParaBackTransparent`
`ParaIsHyphenation, ParaHyphenationMaxLeadingChars/MaxTrailingChars/MaxHyphens`
`Left/Right/Top/BottomBorder, Left/Right/Top/BottomBorderDistance, BorerDistance`
`DropCapFormat, DropCapWholeWord, DropCapCharStyleName`
`PageDescName, PageNumberOffset`
`HyperLinkURL/Name/Target, UnvisitedCharStyleName, VisitedCharStyleName`
`CharEscapement, CharAutoEscapement, CharEscapementHeight`
`CharFontName, CharFontStyleName, CharFontFamily, CharFontPitch`
`CharStrikeOut, CharCrossedOut`
`CharUnderline, CharUnderlineColor, CharUnderlineHasColor`
`CharCombineIsOn, CharCombinePrefix, CharCombineSuffix`
`RubyText, RubyAdjust, RubyCharStyleName, RubyIsAbove`

7.3.3 Navigating

Cursors

The text *model* cursor allows for free navigation over the model by character, words, sentences, or paragraphs. There can be several model cursors at the same time. Model cursor creation, movement and usage is discussed in the section *7.3.1 Text Documents - Working with Text Documents - Word Processing* . The text model cursors are `com.sun.star.text.TextCursor` services that are based on the interface `com.sun.star.text.XTextCursor`, which is based on `com.sun.star.text.XTextRange`.

The text *view* cursor enables the user to travel over the document in the view by character, line, screen page and document page. There is only one text view cursor. Certain information about the current layout, such as the number of lines and page number must be retrieved at the view cursor. The chapter *7.5 Text Documents - Text Document Controller* below discusses the view cursor in detail. The text view cursor is a `com.sun.star.text.TextViewCursor` service that includes `com.sun.star.text.TextLayoutCursor`.

Locating Text Contents

The text document model has suppliers that yield all text contents in a document as collections. To find a particular text content, such as bookmarks or text fields, use the appropriate supplier interface. The following supplier interfaces are available at the model:

Supplier interfaces	**Methods**
`XTextTablesSupplier`	`Com.sun.star.container.XNameAccess` **`getTextTables`**`()`
`XTextFramesSupplier`	`com.sun.star.container.XNameAccess` **`getTextFrames`**`()`
`XTextGraphicObjectsSupplier`	`com.sun.star.container.XNameAccess` **`getGraphicObjects`**`()`
`XTextEmbeddedObjectsSupplier`	`com.sun.star.container.XNameAccess` **`getEmbeddedObjects`**`()`
`XTextFieldsSupplier`	`com.sun.star.container.XEnumerationAccess` **`getTextFields`**`()` `com.sun.star.container.XNameAccess` **`getTextFieldMasters`**`()`
`XBookmarksSupplier`	`com.sun.star.container.XNameAccess` **`getBookmarks`**`()`
`XReferenceMarksSupplier`	`com.sun.star.container.XNameAccess` **`getReferenceMarks`**`()`
`XFootnotesSupplier`	`com.sun.star.container.XIndexAccess` **`getFootnotes`**`()` `com.sun.star.beans.XPropertySet` **`getFootnoteSettings`**`()`
`XEndnotesSupplier`	`com.sun.star.container.XIndexAccess` **`getEndnotes`**`()` `com.sun.star.beans.XPropertySet` **`getEndnoteSettings`**`()`
`XTextSectionsSupplier`	`com.sun.star.container.XNameAccess` **`getTextSections`**`()`
`XDocumentIndexesSupplier`	`com.sun.star.container.XIndexAccess` **`getDocumentIndexes`**`()`
`XRedlinesSupplier`	`com.sun.star.container.XEnumerationAccess` **`getRedlines`**`()`

You can work with text content directly, set properties and use its interfaces, or find out where it is and do an action at the text content location in the text. To find out where a text content is located call the `getAnchor()` method at the interface `com.sun.star.text.XTextContent`, which every text content must support.

In addition, text contents located at the current text cursor position or the content where the cursor is currently located are provided in the `PropertySet` of the cursor. The corresponding cursor properties are:

- DocumentIndexMark
- TextField
- ReferenceMark
- Footnote
- Endnote
- DocumentIndex
- TextTable
- TextFrame
- Cell
- TextSection

Search and Replace

The writer model supports the interface `com.sun.star.util.XReplaceable` that inherits from the interface `com.sun.star.util.XSearchable` for searching and replacing in text. It contains the following methods:

```
com::sun::star::util::util.XSearchDescriptor createSearchDescriptor()
com::sun::star::util::XReplaceDescriptor createReplaceDescriptor()

com::sun::star::uno::XInterface findFirst( [in] com::sun::star::util::XSearchDescriptor xDesc)
com::sun::star::uno::XInterface findNext( [in] com::sun::star::uno::XInterface xStartAt,
                                          [in] com::sun::star::util::XSearchDescriptor xDesc)
com::sun::star::container::XIndexAccess findAll( [in] com::sun::star::util::XSearchDescriptor xDesc)

long replaceAll( [in] com::sun::star::util::XSearchDescriptor xDesc)
```

To search or replace text, first create a descriptor service using `createSearchDescriptor()` or `createReplaceDescriptor()`. You receive a service that supports the interface `com.sun.star.util.XPropertyReplace` with methods to describe what you are searching for, what you want to replace with and what attributes you are looking for. It is described in detail below.

Pass in this descriptor to the methods `findFirst()`, `findNext()`, `findAll()` or `replaceAll()`.

The methods `findFirst()` and `findNext()` return a `com.sun.star.uno.XInterface` pointing to an object that contains the found item. If the search is not successful, a null reference to an `XInterface` is returned, that is, if you try to query other interfaces from it, `null` is returned. The method `findAll()` returns a `com.sun.star.container.XIndexAccess` containing one or more `com.sun.star.uno.XInterface` pointing to the found text ranges or if they failed an empty interface. The method `replaceAll()` returns the number of replaced occurrences only.

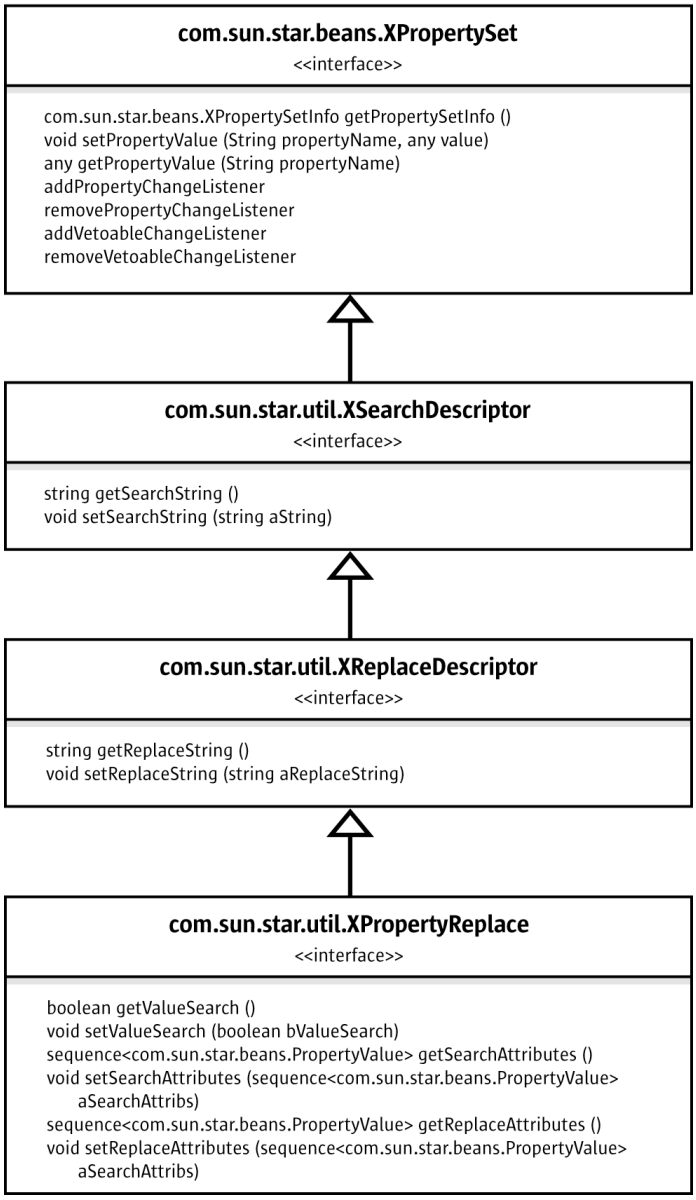


Illustration 74: XPropertyReplace

The interface `com.sun.star.util.XPropertyReplace` is required to describe your search. It is a powerful interface and inherits from `XReplaceDescriptor`, `XSearchDescriptor` and `XPropertySet`.

The target of your search is described by a string containing a search text or a style name using `setSearchString()`. Correspondingly, provide the text string or style name that should replace the found occurrence of the search target to the `XReplaceDescriptor` using `setReplaceString()`. Refine the search mode through the properties included in the service `com.sun.star.util.SearchDescriptor`:

Properties of `com.sun.star.util.SearchDescriptor`	
`SearchBackwards`	`boolean` — Search backward
`SearchCaseSensitive`	`boolean` —Search is case sensitive.

Properties of com.sun.star.util.SearchDescriptor	
SearchRegularExpression	boolean —Search interpreting the search string as a regular expression.
SearchSimilarity	boolean — Use similarity search using the four following options:
SearchSimilarityAdd	short — Determines the number of characters the word in the document may be longer than the search string for it to remain valid.
SearchSimilarityExchange	short —Determines how many characters in the search term can be exchanged.
SearchSimilarityRelax	boolean — If true, the values of added, exchanged, and removed characters are combined The search term is then found if the word in the document can be generated through any combination of these three conditions.
SearchSimilarityRemove	short — Determines how many characters the word in the document may be shorter than the search string for it to remain valid. The characters may be removed from the word at any position.
SearchStyles	boolean —Determines if the search and replace string should be interpreted as paragraph style names. Note that the Display Name of the style has to be used.
SearchWords	boolean —Determines if the search should find complete words only.

In XPropertyReplace, the methods to get and set search attributes, and replace attributes allow the attributes to search for to be defined and the attributes to insert instead of the existing attributes. All of these methods expect a sequence of com.sun.star.beans.PropertyValue structs.

Any properties contained in the services com.sun.star.style.CharacterProperties, com.sun.star.style.CharacterPropertiesAsian and com.sun.star.style.ParagraphProperties can be used for an attribute search. If setValueSearch(false) is used, OpenOffice.org checks if an attribute exists, whereas setValueSearch (true) finds specific attribute values. If only searching to see if an attribute exists at all, it is sufficient to pass a PropertyValue struct with the Name field set to the name of the required attribute.

The following code snippet replaces all occurrences of the text "random numbers" by the bold text "replaced numbers" in a given document mxDoc.

```
XReplaceable xReplaceable = (XReplaceable) UnoRuntime.queryInterface(XReplaceable.class, mxDoc);
XReplaceDescriptor xRepDesc = xReplaceable.createReplaceDescriptor();

// set a string to search for
xRepDesc.setSearchString("random numbers");

// set the string to be inserted
xRepDesc.setReplaceString("replaced numbers");

// create an array of one property value for a CharWeight property
PropertyValue[] aReplaceArgs = new PropertyValue[1];

// create PropertyValue struct
aReplaceArgs[0] = new PropertyValue();
// CharWeight should be bold
aReplaceArgs[0].Name = "CharWeight";
aReplaceArgs[0].Value = new Float(com.sun.star.awt.FontWeight.BOLD);

// set our sequence with one property value as ReplaceAttribute
XPropertyReplace xPropRepl = (XPropertyReplace) UnoRuntime.queryInterface(
        XPropertyReplace.class, xRepDesc);
xPropRepl.setReplaceAttributes(aReplaceArgs);
```

```
// replace
long nResult = xReplaceable.replaceAll(xRepDesc);
```

7.3.4 Tables

Table Architecture

OpenOffice.org text tables consist of rows, rows consist of one or more cells, and cells can contain text or rows. There is no logical concept of columns. From the API's perspective, a table acts as if it had columns, as long as there are no split or merged cells.

Cells in a row are counted alphabetically starting from A, where rows are counted numerically, starting from 1. This results in a cell-row addressing pattern, where the cell letter is denoted first (A-Zff.), followed by the row number (1ff.):

A1	B1	C1	D1
A2	B2	C2	D2
A3	B3	C3	D3
A4	B4	C4	D4

When a cell is *split* vertically, the new cell gets the letter of the former right-hand-side neighbor cell and the former neighbor cell gets the next letter in the alphabet. Consider the example table below: B2 was split vertically, a new cell C2 is inserted and the former C2 became D2, D2 became E2, and so forth.

When cells are *merged* vertically, the resulting cell counts as one cell and gets one letter. The neighbor cell to the right gets the subsequent letter. B4 in the table below shows this. The former B4 and C4 have been merged, so the former D4 could become C4. The cell name D4 is no longer required.

As shown, there is no way to address a column C anymore, for the cells C1 to C4 no longer form a column:

A1	B1		C1	D1
A2	B2 vertically split in two	C2 newly inserted	D2	E2
A3	B3		C3	D3
A4	B4 merged with C4			C4

When cells are split horizontally, OpenOffice.org simply inserts as many rows into the cell as required.

In our example table, we continued by splitting C2 first horizontally and then vertically so that there is a range of four cells.

The writer treats the content of C2 as two rows and starts counting cells within rows. To address the new cells, it extends the original cell name C2 by new addresses following the cell-row pattern. The upper row gets row number 1 and the first cell in the row gets cell number 1, resulting in the cell address C2.1.1, where the latter 1 indicates the row and the former 1 indicates the first cell in the row. The right neighbor of C2.1.1 is C2.2.1. The subaddress 2.1 means the second cell in the first row.

A1	B1			C1	D1
A2	B2 vertically split in two	C2.1.1	C2.2.1	D2	E2
		C2.1.2	C2.2.2		
A3	B3			C3	D3
A4	B4 merged with C4				C4

The cell-row pattern is used for all further subaddressing as the cells are split and merged. The cell addresses can change radically depending on the table structure generated by OpenOffice.org. The next table shows what happens when E2 is merged with D3. The table is reorganized, so that it has three rows instead of four. The second row contains *two* cells, A2 and B2 (sic!). The cell A2 has two rows, as shown from the cell subaddresses: The upper row consists of four cells, namely A2.1.1 through A2.4.1, whereas the lower row consists of the three cells A2.1.2 through A2.3.2.

The cell range C2.1.1:C2.2.2 that was formerly contained in cell C2 is now in cell A2.3.1 that denotes the third cell in the first row of A2. Within the address of the cell A2.3.1, OpenOffice.org has started a new subaddressing level using the cell-row pattern again.

A1	B1			C1	D1
A2.1.1	A2.2.1	A2.3.1.1.1	A2.3.1.2.1	A2.4.1	Former E2 merged with former D3 Becomes B2!
		A2.3.1.1.2	A2.3.1.2.2		
A2.1.2	A2.2.2			A2.3.2	
A3	B3				C3

Cell addresses can become complicated. The cell address can be looked up in the user interface. Set the GUI text cursor in the desired cell and observe the lower-right corner of the status bar in the text document.

Remember that there are only "columns" in a text table, as long as there are no split or merged cells.

Text tables support the service `com.sun.star.text.TextTable`, which includes the service `com.sun.star.text.TextContent`:

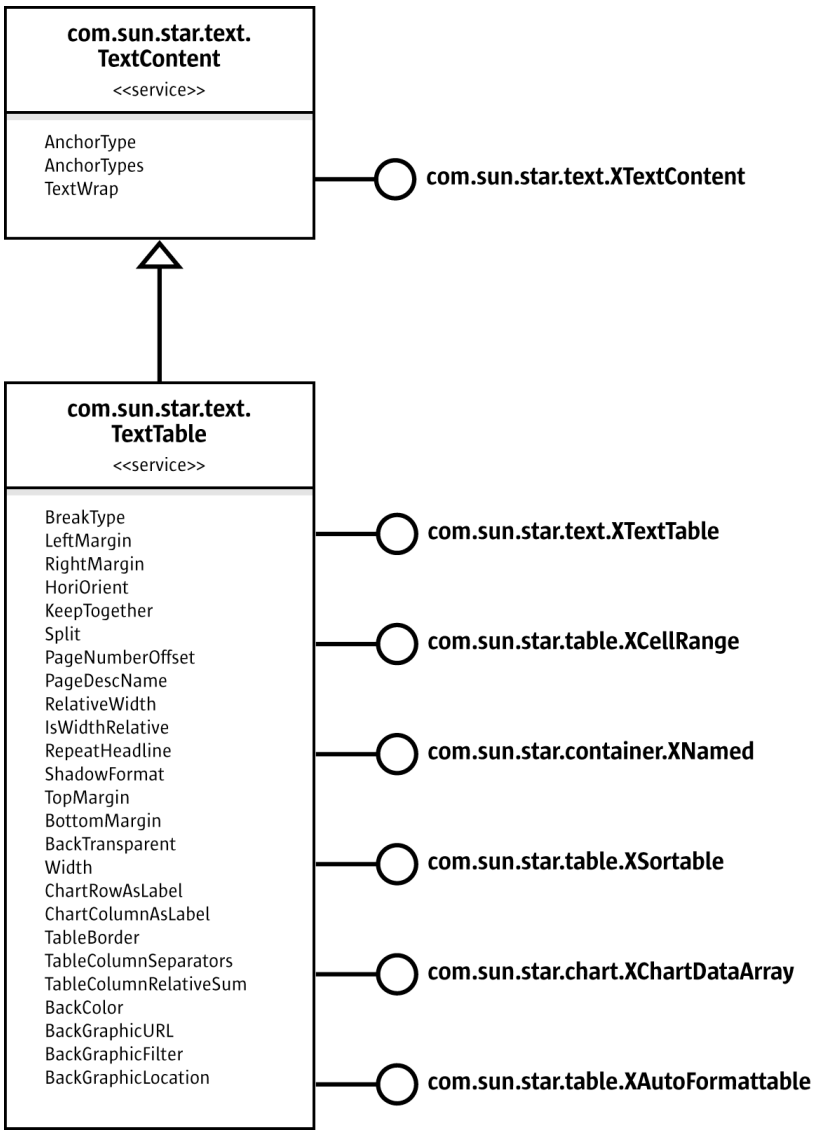


Illustration 75 Service com.sun.star.text.TextTable

The service `com.sun.star.text.TextTable` offers access to table cells in two different ways::

- Yields named table cells which are organized in rows and columns.
- Provides a table cursor to travel through the table cells and alter the cell properties.

These aspects are reflected in the interface `com.sun.star.text.XTextTable` which inherits from `com.sun.star.text.XTextContent`. It can be seen as a rectangular range of cells defined by numeric column indexes, as described by `com.sun.star.table.XCellRange`. This aspect makes text tables compatible with spreadsheet tables. Also, text tables have a name, can be sorted, charts can be based on them, and predefined formats can be applied to the tables. The latter aspects are covered by the interfaces `com.sun.star.container.XNamed`, `com.sun.star.util.XSortable`, `com.sun.star.chart.XChartDataArray` and `com.sun.star.table.XAutoFormattable`.

The usage of these interfaces and the properties of the TextTable service are discussed below.

Named Table Cells in Rows, Columns and the Table Cursor

The interface `XTextTable` introduces the following methods to initialize a table, work with table cells, rows and columns, and create a table cursor:

```
void initialize( [in] long nRows,  [in] long nColumns)

sequence< string > getCellNames()
com::sun::star::table::XCell getCellByName( [in] string aCellName)

com::sun::star::table::XTableRows getRows()
com::sun::star::table::XTableColumns getColumns()

com::sun::star::text::XTextTableCursor createCursorByCellName( [in] string aCellName)
```

The method `initialize()` sets the number of rows and columns prior to inserting the table into the text. Non-initialized tables default to two rows and two columns.

The method `getCellNames()` returns a sequence of strings containing the names of all cells in the table in A1[.1.1] notation.

The method `getCellByName()` expects a cell name in A1[.1.1] notation, and returns a cell object that is a `com.sun.star.table.XCell` and a `com.sun.star.text.XText`. The advantage of `getCellByName()` is its ability to retrieve cells even in tables with split or merged cells.

The method `getRows()` returns a table row container supporting `com.sun.star.table.XTableRows` that is a `com.sun.star.container.XIndexAccess`, and introduces the following methods to insert an arbitrary number of table rows below a given row index position and remove rows from a certain position:

```
void insertByIndex ( [in] long nIndex,  [in] long nCount)
void removeByIndex ( [in] long nIndex,  [in] long nCount)
```

The following table shows which `XTableRows` methods work under which circumstances.

Method in `com.sun.star.table.XTableRows`	**In Simple table**	**In Complex Table**
`getElementType()`	X	X
`hasElements()`	X	X
`getByIndex()`	X	X
`getCount()`	X	X
`insertByIndex()`	X	-
`removeByIndex()`	X	-

Every row returned by `getRows()` supports the service `com.sun.star.text.TextTableRow`, that is, it is a `com.sun.star.beans.XPropertySet` which features these properties:

Properties of `com.sun.star.text.TextTableRow`	
`RowBackColor`	`long` — Specifies the color of the background in 0xAARRGGBB notation.
`BackTransparent`	`boolean` — If `true`, the background color value in "`BackColor`" is not visible.
`VertOrient`	The vertical orientation of the text inside of the table cells in this row, `com.sun.star.text.VertOrientation` VertOrient can only be read in Build 641. There is no way to set the vertical orientation of text table cell contents. It defaults to top for strings and bottom for numeric values.

Properties of com.sun.star.text.TextTableRow	
BackGraphicURL	string — Contains the URL of a background graphic.
BackGraphicFilter	string — Contains the name of the file filter of a background graphic.
BackGraphicLocation	com.sun.star.style.GraphicLocation. Determines the position of the background graphic.
TableColumnSeparator	Defines the column width and its merging behavior. It contains a sequence of com.sun.star.text.TableColumnSeparator structs with the fields Position and IsVisible. The value of Position is relative to the table property com.sun.star.text.TextTable:TableColumnRelativeSum. IsVisible refers to merged cells where the separator becomes invisible.
Height	long — Contains the height of the table row.
IsAutoHeight	boolean — If the value of this property is true , the height of the table row depends on the content of the table cells.

The method getColumns() is similar to getRows(), but restrictions apply. It returns a table column container supporting com.sun.star.table.XTableColumns that is a com.sun.star.container.XIndexAccess and introduces the following methods to insert an arbitrary number of table columns behind a given column index position and remove columns from a certain position:

```
void insertByIndex( [in] long nIndex,  [in] long nCount)
void removeByIndex( [in] long nIndex,  [in] long nCount)
```

The following table shows which XTableColumns methods work in which situation.

Method in com.sun.star.table.XTableColumns	In Simple Table	In Complex Table
getElementType()	X	X
hasElements()	X	X
getByIndex()	X (but returned object supports XInterface only)	-
getCount()	X	-
insertByIndex()	X	-
removeByIndex()	X	-

The method createCursorByCellName() creates a text table cursor that can select a cell range in the table, merge or split cells, and read and write cell properties of the selected cell range. It is a com.sun.star.text.TextTableCursor service with the interfaces com.sun.star.text.XTextTableCursor and com.sun.star.beans.XPropertySet.

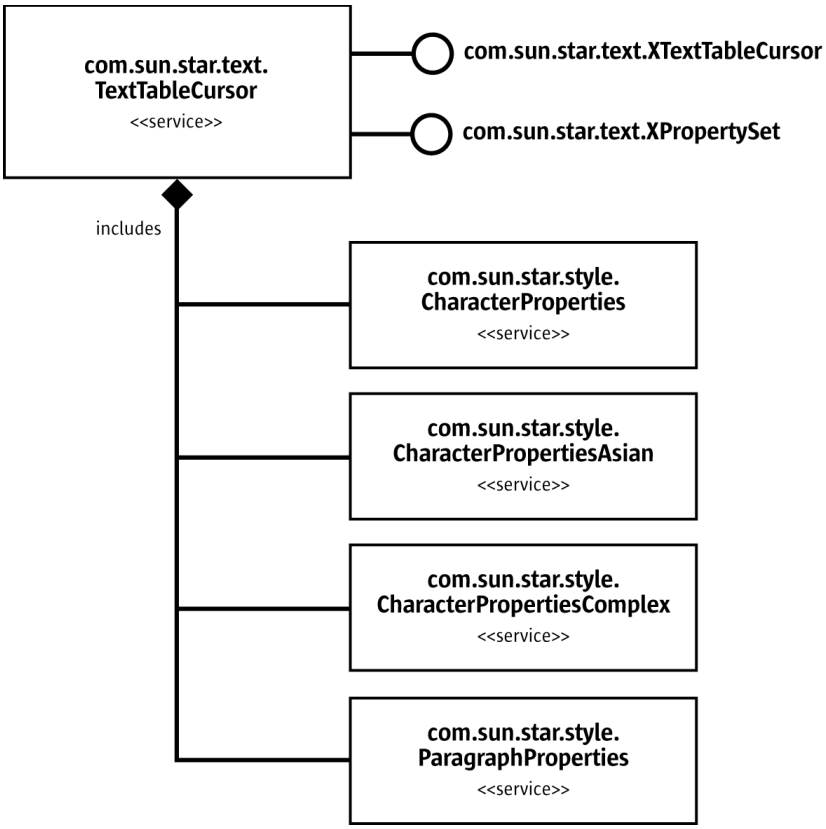


Illustration 76 com.sun.star.text.TextTableCursor

These are the methods contained in `XTextTableCursor`:

```
string getRangeName()

boolean goLeft( [in] short nCount,  [in] boolean bExpand)
boolean goRight( [in] short nCount,  [in] boolean bExpand)
boolean goUp( [in] short nCount,  [in] boolean bExpand)
boolean goDown( [in] short nCount,  [in] boolean bExpand)

void gotoStart( [in] boolean bExpand)
void gotoEnd( [in] boolean bExpand)
boolean gotoCellByName( [in] string aCellName,  [in] boolean bExpand)

boolean mergeRange()
boolean splitRange( [in] short Count,  [in] boolean Horizontal)
```

Traveling through the table calls the cursor's `goLeft()`, `goRight()`, `goUp()`, `goDown()`, `gotoStart()`, `gotoEnd()`, and `gotoCellByName()` methods, passing `true` to select cells on the way.

Once a cell range is selected, apply character and paragraph properties to the cells in the range as defined in the services `com.sun.star.style.CharacterProperties`, `com.sun.star.style.CharacterPropertiesAsian`, `com.sun.star.style.CharacterPropertiesComplex` and `com.sun.star.style.ParagraphProperties`. Moreover, split and merge cells using the text table cursor. An example is provided below.

Indexed Cells and Cell Ranges

The interface `com.sun.star.table.XCellRange` provides access to cells using their row and column index as position, and to create sub ranges of tables:

```
com::sun::star::table::XCell getCellByPosition( [in] long nColumn,  [in] long nRow)
```

```
com::sun::star::table::XCellRange getCellRangeByPosition( [in] long nLeft,  [in] long nTop,
                                                          [in] long nRight, [in] long nBottom)
com::sun::star::table::XCellRange getCellRangeByName( [in] string aRange)
```

The method `getCellByPosition()` returns a cell object supporting the interfaces `com.sun.star.table.XCell` and `com.sun.star.text.XText`. To find the cell the name is internally created from the position using the naming scheme described above and returns this cell if it exists. Calling `getCellByPosition(1, 1)` in the table at the beginning of this chapter returns the cell "B2" .

The methods `getCellRangeByPosition()` and `getCellRangeByName()` return a range object that is described below. The name of the range is created with the top-left cell and bottom-right cell of the table separated by a colon : as in `A1:B4`. Both methods fail when the structure of the table contains merged or split cells.

Table Naming, Sorting, Charting and Autoformatting

Each table has a unique name that can be read and written using the interface `com.sun.star.container.XNamed`.

A text table is a `com.sun.star.util.XSortable`. Its method `createSortDescriptor()` returns a sequence of `com.sun.star.beans.PropertyValue` structs that provides the elements as described in the service `com.sun.star.text.TextSortDescriptor`. The method `sort()` sorts the table content by the given parameters.

The interface `com.sun.star.chart.XChartDataArray` is used to connect a table or a range inside of a table to a chart. It reads and writes the values of a range, and sets the column and row labels. The inherited interface `com.sun.star.chart.XChartData` enables the chart to connect listeners to be notified when changes to the values of a table are made. For details about charting, refer to chapter *10 Charts*.

The interface `com.sun.star.table.XAutoFormattable` provides in its method `autoFormat()` a method to format the table using a predefined table format. To access the available auto formats, the service `com.sun.star.sheet.TableAutoFormats` has to be accessed. For details, refer to chapter *8.3.2 Spreadsheet Documents - Working with Spreadsheets - Formatting - Table Auto Formats*.

Text Table Properties

The text table supports the properties described in the service `com.sun.star.text.TextTable`:

Properties of `com.sun.star.text.TextTable`	
`BackColor`	`long` — Contains the color of the table background.
`BackGraphicFilter`	`string` — Contains the name of the file filter for the background graphic.
`BackGraphicLocation`	`com.sun.star.style.GraphicLocation`. Determines the position of the background graphic.
`BackGraphicURL`	`string` — Contains the URL for the background graphic.
`BackTransparent`	`boolean` — Determines if the background color is transparent.
`BottomMargin`	`long` — Determines the bottom margin.
`BreakType`	`com.sun.star.style.BreakType`. Determines the type of break that is applied at the beginning of the table.
`ChartColumnAsLabel`	`boolean` — Determines if the first column of the table should be treated as axis labels when a chart is to be created.

Properties of `com.sun.star.text.TextTable`	
`ChartRowAsLabel`	`boolean` — Determines if the first row of the table should be treated as axis labels when a chart is to be created.
`HoriOrient`	`short` — Contains the horizontal orientation according to `com.sun.star.text.HoriOrientation`.
`IsWidthRelative`	`boolean` — Determines if the value of the relative width is valid.
`KeepTogether`	`boolean` — Setting this property to `true` prevents page or column breaks between this table and the following paragraph or text table.
`LeftMargin`	`long` — Contains the left margin of the table.
`PageDescName`	`string` — If this property is set, it creates a page break before the table and assigns the value as the name of the new page style sheet to use.
`PageNumberOffset`	`short` — If a page break property is set at the table, this property contains the new value for the page number.
`RelativeWidth`	`short` — Determines the width of the table relative to its environment.
`RepeatHeadline`	`boolean` — Determines if the first row of the table is repeated on every new page.
`RightMargin`	`long` — Contains the right margin of the table.
`ShadowFormat`	`struct com.sun.star.table.ShadowFormat` determines the type, color and size of the shadow.
`Split`	`boolean` — Setting this property to `false` prevents the table from getting spread on two pages.
`TableBorder`	`struct com.sun.star.table.TableBorder`. Contains the description of the table borders.
`TableColumnRelativeSum`	`short` — Contains the sum of the column width values used in TableColumnSeparators.
`TableColumnSeparators`	`sequence < com.sun.star.text.TableColumnSeparator >`. Defines the column width and its merging behavior. It contains a sequence of `com.sun.star.text.TableColumnSeparator` structs with the fields Position and IsVisible. The value of Position is relative to the table property `com.sun.star.text.TextTable:TableColumnRelativeSum`. IsVisible refers to merged cells where the separator becomes invisible. In tables with merged or split cells, the sequence TableColumnSeparators is empty.
`TopMargin`	`long` — Determines the top margin.
`Width`	`long` — Contains the absolute table width.

Inserting Tables

To create and insert a new text table, a five-step procedure must be followed:

1. Get the service manager of the text document, querying the document's factory interface `com.sun.star.lang.XMultiServiceFactory`.
2. Order a new text table from the factory by its service name "`com.sun.star.text.TextTable`", using the factory method `createInstance()`.
3. From the object received, query the `com.sun.star.text.XTextTable` interface that inherits from `com.sun.star.text.XTextContent`.

4. If necessary, initialize the table with the number of rows and columns. For this purpose, XTextTable offers the `initialize()` method.

5. Insert the table into the text using the `insertTextContent()` method at its `com.sun.star.text.XText` interface. The method `insertTextContent()` expects an XTextContent to insert. Since `XTextTable` inherits from `XTextContent`, pass the `XTextTable` interface retrieved previously.

You are now ready to get cells, fill in text, values and formulas and set the table and cell properties as needed.

In the following code sample, there is a small helper function to put random numbers between -1000 and 1000 into the table to demonstrate formulas: (Text/TextDocuments.java)

```
/** This method returns a random double which isn't too high or too low
 */
protected double getRandomDouble()
{
    return ((maRandom.nextInt() % 1000) * maRandom.nextDouble());
}
```

The following helper function inserts a string into a cell known by its name and sets its text color to white: (Text/TextDocuments.java)

```
/** This method sets the text colour of the cell refered to by sCellName to white and inserts
    the string sText in it
 */
public static void insertIntoCell(String sCellName, String sText, XTextTable xTable) {
    // Access the XText interface of the cell referred to by sCellName
    XText xCellText = (XText) UnoRuntime.queryInterface(
        XText.class, xTable.getCellByName(sCellName));

    // create a text cursor from the cells XText interface
    XTextCursor xCellCursor = xCellText.createTextCursor();

    // Get the property set of the cell's TextCursor
    XPropertySet xCellCursorProps = (XPropertySet)UnoRuntime.queryInterface(
        XPropertySet.class, xCellCursor);

    try {
        // Set the colour of the text to white
        xCellCursorProps.setPropertyValue("CharColor", new Integer(16777215));
    } catch (Exception e) {
        e.printStackTrace(System.out);
    }

    // Set the text in the cell to sText
    xCellText.setString(sText);
}
```

Using the above helper functions, create a text table and insert it into the text document. (Text/TextDocuments.java)

```
/** This method shows how to create and insert a text table, as well as insert text and formulae
                into the cells of the table
 */
protected void TextTableExample ()
{
        try
        {
                // Create a new table from the document's factory
                XTextTable xTable = (XTextTable) UnoRuntime.queryInterface(
                        XTextTable.class, mxDocFactory .createInstance(
                                "com.sun.star.text.TextTable" ) );

                // Specify that we want the table to have 4 rows and 4 columns
                xTable.initialize( 4, 4 );

                // Insert the table into the document
                mxDocText.insertTextContent( mxDocCursor, xTable, false);
                // Get an XIndexAccess of the table rows
                XIndexAccess xRows = xTable.getRows();

                // Access the property set of the first row (properties listed in service description:
                // com.sun.star.text.TextTableRow)
                XPropertySet xRow = (XPropertySet) UnoRuntime.queryInterface(
                        XPropertySet.class, xRows.getByIndex ( 0 ) );
                // If BackTransparant is false, then the background color is visible
                xRow.setPropertyValue( "BackTransparent", new Boolean(false));
```

```
                // Specify the color of the background to be dark blue
                xRow.setPropertyValue( "BackColor", new Integer(6710932));

                // Access the property set of the whole table
                XPropertySet xTableProps = (XPropertySet)UnoRuntime.queryInterface(
                        XPropertySet.class, xTable );
                // We want visible background colors
                xTableProps.setPropertyValue( "BackTransparent", new Boolean(false));
                // Set the background colour to light blue
                xTableProps.setPropertyValue( "BackColor", new Integer(13421823));

                // set the text (and text colour) of all the cells in the first row of the table
                insertIntoCell( "A1", "First Column", xTable );
                insertIntoCell( "B1", "Second Column", xTable );
                insertIntoCell( "C1", "Third Column", xTable );
                insertIntoCell( "D1", "Results", xTable );

                // Insert random numbers into the first this three cells of each
                // remaining row
                xTable.getCellByName( "A2" ).setValue( getRandomDouble() );
                xTable.getCellByName( "B2" ).setValue( getRandomDouble() );
                xTable.getCellByName( "C2" ).setValue( getRandomDouble() );

                xTable.getCellByName( "A3" ).setValue( getRandomDouble() );
                xTable.getCellByName( "B3" ).setValue( getRandomDouble() );
                xTable.getCellByName( "C3" ).setValue( getRandomDouble() );

                xTable.getCellByName( "A4" ).setValue( getRandomDouble() );
                xTable.getCellByName( "B4" ).setValue( getRandomDouble() );
                xTable.getCellByName( "C4" ).setValue( getRandomDouble() );

                // Set the last cell in each row to be a formula that calculates
                // the sum of the first three cells
                xTable.getCellByName( "D2" ).setFormula( "sum <A2:C2>" );
                xTable.getCellByName( "D3" ).setFormula( "sum <A3:C3>" );
                xTable.getCellByName( "D4" ).setFormula( "sum <A4:C4>" );
        }
        catch (Exception e)
        {
                e.printStackTrace ( System.out );
        }
}
```

The next sample inserts auto text entries into a table, splitting cells during its course. (Text/TextDocuments.java)

```
/** This example demonstrates the use of the AutoTextContainer, AutoTextGroup and AutoTextEntry services
    and shows how to create, insert and modify auto text blocks
 */
protected void AutoTextExample ()
{
        try
        {
                // Go to the end of the document
                mxDocCursor.gotoEnd( false );
                // Insert two paragraphs
                mxDocText.insertControlCharacter ( mxDocCursor,
                        ControlCharacter.PARAGRAPH_BREAK, false );
                mxDocText.insertControlCharacter ( mxDocCursor,
                        ControlCharacter.PARAGRAPH_BREAK, false );
                // Position the cursor in the second paragraph
                XParagraphCursor xParaCursor = (XParagraphCursor) UnoRuntime.queryInterface(
                        XParagraphCursor.class, mxDocCursor );
                xParaCursor.gotoPreviousParagraph ( false );

                // Get an XNameAccess interface to all auto text groups from the document factory
                XNameAccess xContainer = (XNameAccess) UnoRuntime.queryInterface(
                        XNameAccess.class, mxFactory.createInstance (
                                "com.sun.star.text.AutoTextContainer" ) );

                // Create a new table at the document factory
                XTextTable xTable = (XTextTable) UnoRuntime.queryInterface(
                        XTextTable.class, mxDocFactory .createInstance(
                                "com.sun.star.text.TextTable" ) );

                // Store the names of all auto text groups in an array of strings
                String[] aGroupNames = xContainer.getElementNames();

                // Make sure we have at least one group name
                if ( aGroupNames.length > 0 )
                {
                        // initialise the table to have a row for every autotext group
                        //in a single column + one
                        // additional row for a header
                        xTable.initialize( aGroupNames.length+1,1);
```

```
// Access the XPropertySet of the table
XPropertySet xTableProps = (XPropertySet)UnoRuntime.queryInterface(
        XPropertySet.class, xTable );

// We want a visible background
xTableProps.setPropertyValue( "BackTransparent", new Boolean(false));

// We want the background to be light blue
xTableProps.setPropertyValue( "BackColor", new Integer(13421823));

// Inser the table into the document
mxDocText.insertTextContent( mxDocCursor, xTable, false);

// Get an XIndexAccess to all table rows
XIndexAccess xRows = xTable.getRows();

// Get the first row in the table
XPropertySet xRow = (XPropertySet) UnoRuntime.queryInterface(
        XPropertySet.class, xRows.getByIndex ( 0 ) );

// We want the background of the first row to be visible too
xRow.setPropertyValue( "BackTransparent", new Boolean(false));

// And let's make it dark blue
xRow.setPropertyValue( "BackColor", new Integer(6710932));

// Put a description of the table contents into the first cell
insertIntoCell( "A1", "AutoText Groups", xTable);

// Create a table cursor pointing at the second cell in the first column
XTextTableCursor xTableCursor = xTable.createCursorByCellName ( "A2" );

// Loop over the group names
for ( int i = 0 ; i < aGroupNames.length ; i ++ )
{
        // Get the name of the current cell
        String sCellName = xTableCursor.getRangeName ();

        // Get the XText interface of the current cell
        XText xCellText = (XText) UnoRuntime.queryInterface (
                XText.class, xTable.getCellByName ( sCellName ) );

        // Set the cell contents of the current cell to be
        //the name of the of an autotext group
        xCellText.setString ( aGroupNames[i] );

        // Access the autotext gruop with this name
        XAutoTextGroup xGroup = ( XAutoTextGroup ) UnoRuntime.queryInterface (
                XAutoTextGroup.class,xContainer.getByName(aGroupNames[i]));

        // Get the titles of each autotext block in this group
        String [] aBlockNames = xGroup.getTitles();

        // Make sure that the autotext group contains at least one block
        if ( aBlockNames.length > 0 )
        {
                // Split the current cell vertically into two seperate cells
                xTableCursor.splitRange ( (short) 1, false );

                // Put the cursor in the newly created right hand cell
                // and select it
                xTableCursor.goRight ( (short) 1, false );

                // Split this cell horizontally to make a seperate cell
                // for each Autotext block
                if ( ( aBlockNames.length -1 ) > 0 )
                        xTableCursor.splitRange (
                                (short) (aBlockNames.length - 1), true );

                // loop over the block names
                for ( int j = 0 ; j < aBlockNames.length ; j ++ )
                {
                        // Get the XText interface of the current cell
                        xCellText = (XText) UnoRuntime.queryInterface (
                                XText.class, xTable.getCellByName (
                                        xTableCursor.getRangeName() ) );

                        // Set the text contents of the current cell to the
                        // title of an Autotext block
                        xCellText.setString ( aBlockNames[j] );

                        // Move the cursor down one cell
                        xTableCursor.goDown( (short)1, false);
                }
        }
        // Go back to the cell we originally split
        xTableCursor.gotoCellByName ( sCellName, false );
```

```
                                // Go down one cell
                                xTableCursor.goDown( (short)1, false);
                        }

                        XAutoTextGroup xGroup;
                        String [] aBlockNames;

                        // Add a depth so that we only generate 200 numbers before
                        // giving up on finding a random autotext group that contains autotext blocks
                        int nDepth = 0;
                        do
                        {
                                // Generate a random, positive number which is lower than
                                // the number of autotext groups
                                int nRandom = Math.abs ( maRandom.nextInt() % aGroupNames.length );

                                // Get the autotext group at this name
                                xGroup  = ( XAutoTextGroup ) UnoRuntime.queryInterface (
                                        XAutoTextGroup.class, xContainer.getByName (
                                                aGroupNames[ nRandom ] ) );

                                // Fill our string array with the names of all the blocks in this
                                // group
                                aBlockNames = xGroup.getElementNames();

                                // increment our depth counter
                                ++nDepth;
                        }
                        while ( nDepth < 200 && aBlockNames.length == 0 );
                        // If we managed to find a group containg blocks...
                        if ( aBlockNames.length > 0 )
                        {
                                // Pick a random block in this group and get it's
                                // XAutoTextEntry interface
                                int nRandom = Math.abs ( maRandom.nextInt()
                                                        % aBlockNames.length );
                                XAutoTextEntry xEntry = ( XAutoTextEntry )
                                         UnoRuntime.queryInterface (
                                                XAutoTextEntry.class, xGroup.getByName (
                                                        aBlockNames[ nRandom ] ) );
                                // insert the modified autotext block at the end of the document
                                xEntry.applyTo ( mxDocCursor );

                                // Get the titles of all text blocks in this AutoText group
                                String [] aBlockTitles = xGroup.getTitles();

                                // Get the XNamed interface of the autotext group
                                XNamed xGroupNamed = ( XNamed ) UnoRuntime.queryInterface (
                                         XNamed.class, xGroup );

                                // Output the short cut and title of the random block
                                //and the name of the group it's from
                                System.out.println ( "Inserted the Autotext '" + aBlockTitles[nRandom]
                                        + "', shortcut '" + aBlockNames[nRandom] + "' from group '"
                                        + xGroupNamed.getName() );
                        }
                }

                // Go to the end of the document
                mxDocCursor.gotoEnd( false );
                // Insert new paragraph
                mxDocText.insertControlCharacter (
                        mxDocCursor, ControlCharacter.PARAGRAPH_BREAK, false );

                // Position cursor in new paragraph
                xParaCursor.gotoPreviousParagraph ( false );

                // Insert a string in the new paragraph
                mxDocText.insertString ( mxDocCursor, "Some text for a new autotext block", false );

                // Go to the end of the document
                mxDocCursor.gotoEnd( false );
        }
        catch (Exception e)
        {
                e.printStackTrace ( System.out );
        }
}
```

Accessing Existing Tables

To access the tables contained in a text document, the text document model supports the interface `com.sun.star.text.XTextTablesSupplier` with one single method `getTextTables()`. It returns a `com.sun.star.text.TextTables` service, which is a named and indexed collection, that is, tables are retrieved using `com.sun.star.container.XNameAccess` or `com.sun.star.container.XIndexAccess`.

The following snippet iterates over the text tables in a given text document object `mxDoc` and colors them green.

```
import com.sun.star.text.XTextTablesSupplier;
import com.sun.star.container.XNameAccess;
import com.sun.star.container.XIndexAccess;
import com.sun.star.beans.XPropertySet;

...

// first query the XTextTablesSupplier interface from our document
XTextTablesSupplier xTablesSupplier = (XTextTablesSupplier) UnoRuntime.queryInterface(
        XTextTablesSupplier.class, mxDoc );
// get the tables collection
XNameAccess xNamedTables = xTablesSupplier.getTextTables();

// now query the XIndexAccess from the tables collection
XIndexAccess xIndexedTables = (XIndexAccess) UnoRuntime.queryInterface(
        XIndexAccess.class, xNamedTables);

// we need properties
XPropertySet xTableProps = null;

// get the tables
for (int i = 0; i < xIndexedTables.getCount(); i++) {
        Object table = xIndexedTables.getByIndex(i);
        // the properties, please!
        xTableProps = (XPropertySet) UnoRuntime.queryInterface(
                XPropertySet.class, table);

        // color the table light green in format 0xRRGGBB
        xTableProps.setPropertyValue("BackColor", new Integer(0xC8FFB9));
}
```

7.3.5 Text Fields

Text fields are text contents that add a second level of information to text ranges. Usually their appearance fuses together with the surrounding text, but actually the presented text comes from elsewhere. Field commands can insert the current date, page number, total page numbers, a cross-reference to another area of text, the content of certain database fields, and many variables, such as fields with changing values, into the document. There are some fields that contain their own data, where others get the data from an attached field master.

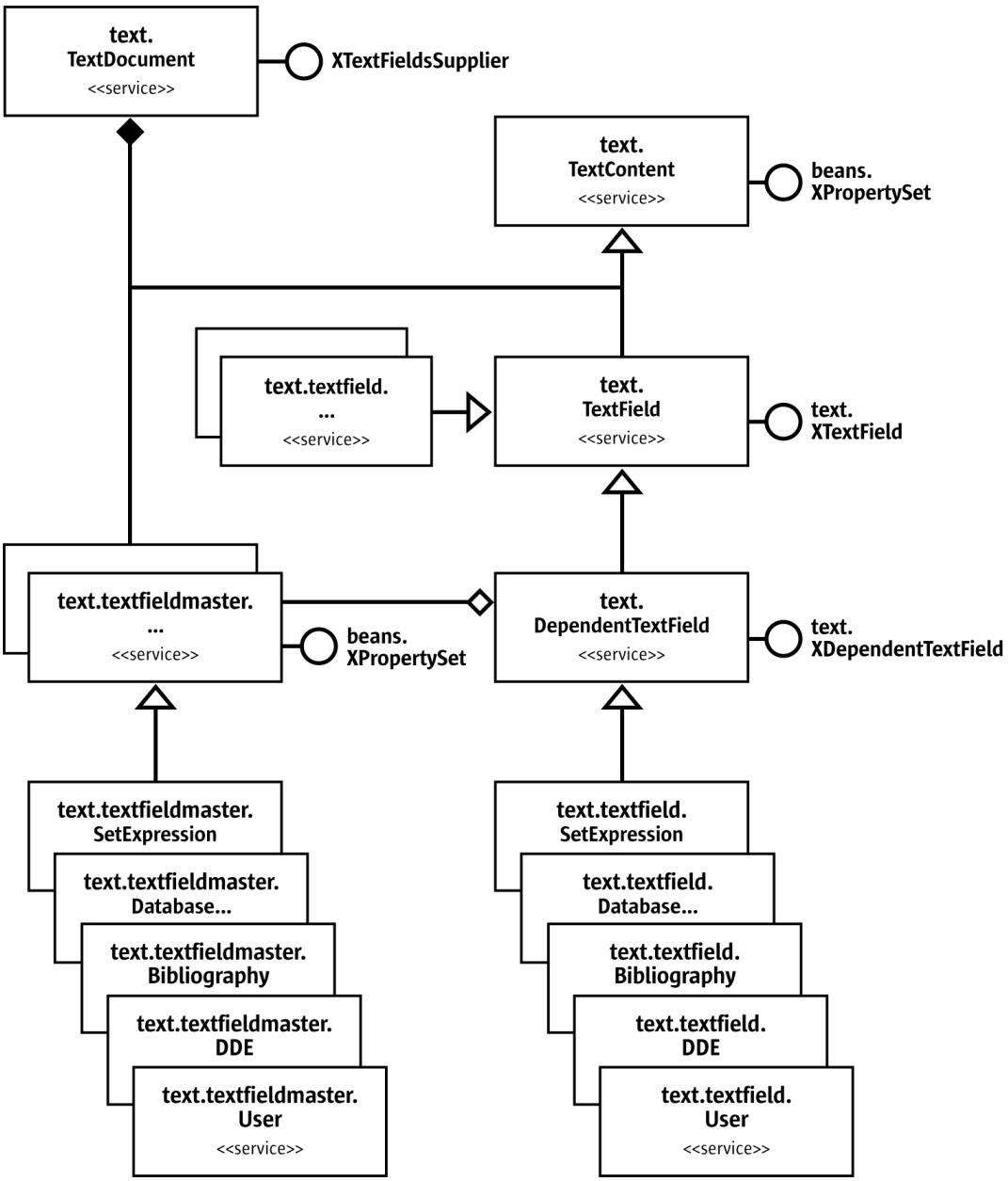


Illustration 77Text Fields and Text Field Masters

Fields are created using the `com.sun.star.lang.XMultiServiceFactory` of the model before inserting them using `insertTextContent()`. The following text field services are available:

Text Field Service Name	Description
`com.sun.star.text.textfield.Annotation`	Annotation created through **Insert – Note.**
`com.sun.star.text.textfield.Author`	Shows the author of the document.

Text Field Service Name	Description
com.sun.star.text.textfield.Bibliography	Bibliographic entry created by **Insert – Indexes and Tables – Bibliography Entry**. The content is the source of the creation of bibliographic indexes. The sequence <PropertyValue> in the property "Fields" contains pairs of the name of the field and its content, such as: Identifier=ABC99 BibliographicType=1 The names of the fields are defined in com.sun.star.text.BibliographyDataField. A bibliographic entry depends on com.sun.star.text.FieldMaster.Bibliography
com.sun.star.text.textfield.Chapter	Show the chapter information.
com.sun.star.text.textfield.CharacterCount	Show the character count of the document.
com.sun.star.text.textfield.CombinedCharacters	Combines up to six characters as one text object that is formatted in two lines.
com.sun.star.text.textfield.ConditionalText	Inserts text depending on a condition.
com.sun.star.text.textfield.Database	The form letter field showing the content from a database. Depends on com.sun.star.text.FieldMaster.Database.
com.sun.star.text.textfield.DatabaseName	Shows the name of a database.
com.sun.star.text.textfield.DatabaseNextSet	Increments the cursor that points to a database selection.
com.sun.star.text.textfield.DatabaseNumberOfSet	Shows the set number of a database cursor.
com.sun.star.text.textfield.DatabaseSetNumber	Databases - Any Record. Sets the number of a database cursor.
com.sun.star.text.textfield.DateTime	Shows a date or time value.
com.sun.star.text.textfield.DDE	Shows the result of a DDE operation. Depends on com.sun.star.text.FieldMaster.DDE.
com.sun.star.text.textfield.docinfo.ChangeAuthor	Shows the name of the author of the last change of the document.
com.sun.star.text.textfield.docinfo.ChangeDateTime	Shows the date and time of the last change of the document.
com.sun.star.text.textfield.docinfo.CreateAuthor	Shows the name of the creator of the document.
com.sun.star.text.textfield.docinfo.CreateDateTime	Shows the date and time of the document creation.
com.sun.star.text.textfield.docinfo.Description	Shows the description contained in the document information.
com.sun.star.text.textfield.docinfo.EditTime	Shows the time of the editing of the document.
com.sun.star.text.textfield.docinfo.Info0	Shows the content of the first user defined info field of the document info.

Text Field Service Name	Description
com.sun.star.text.textfield.docinfo.Info1	Shows the content of the second user defined info field of the document info.
com.sun.star.text.textfield.docinfo.Info2	Shows the content of the third user defined info field of the document info.
com.sun.star.text.textfield.docinfo.Info3	Shows the content of the fourth user defined info field of the document info.
com.sun.star.text.textfield.docinfo.Keywords	Shows the keywords contained in the document info.
com.sun.star.text.textfield.docinfo.PrintAuthor	Shows the name of the author of the last printing.
com.sun.star.text.textfield.docinfo.PrintDateTime	Shows the date and time of the last printing.
com.sun.star.text.textfield.docinfo.Revision	Shows the revision contained in the document info.
com.sun.star.text.textfield.docinfo.Subject	Shows the subject contained in the document info.
com.sun.star.text.textfield.docinfo.Title	Shows the title contained in the document info.
com.sun.star.text.textfield.EmbeddedObjectCount	Shows the number of embedded objects contained in the document.
com.sun.star.text.textfield.ExtendedUser	Shows the user data of the Office user.
com.sun.star.text.textfield.FileName	Shows the file name (URL) of the document.
com.sun.star.text.textfield.GetExpression	Variables – Show Variable. Shows the value set by the previous occurrence of SetExpression.
com.sun.star.text.textfield.GetReference	References – Insert Reference. Shows a reference to a reference mark, bookmark, number range field, footnote or an endnote.
com.sun.star.text.textfield.GraphicObjectCount	Shows the number of graphic object in the document.
com.sun.star.text.textfield.HiddenParagraph	Depending on a condition, the field hides the paragraph it is contained in.
com.sun.star.text.textfield.HiddenText	Depending on a condition the field shows or hides a text.
com.sun.star.text.textfield.Input	The field activates a dialog to input a value that changes a related User field or SetExpression field.
com.sun.star.text.textfield.InputUser	The field activates a dialog to input a string that is displayed by the field. This field is not connected to variables.
com.sun.star.text.textfield.JumpEdit	A placeholder field with an attached interaction to insert text, a text table, text frame, graphic object or an OLE object.
com.sun.star.text.textfield.Macro	A field connected to a macro that is executed on a click to the field. To execute such a macro, use the dispatch (cf. Appendix).
com.sun.star.text.textfield.PageCount	Shows the number of pages of the document.
com.sun.star.text.textfield.PageNumber	Shows the page number (current, previous, next).

Text Field Service Name	Description
com.sun.star.text.textfield.ParagraphCount	Shows the number of paragraphs contained in the document.
com.sun.star.text.textfield.ReferencePageGet	Displays the page number with respect to the reference point, that is determined by the text field ReferencePageSet.
com.sun.star.text.textfield.ReferencePageSet	Inserts a starting point for additional page numbers that can be switched on or off.
com.sun.star.text.textfield.Script	Contains a script or a URL to a script.
com.sun.star.text.textfield.SetExpression	Variables – Set Variable. A variable field. The value is valid until the next occurrence of SetExpression field. The actual value depends on com.sun.star.text.FieldMaster.SetExpression.
com.sun.star.text.textfield.TableCount	Shows the number of text tables of the document.
com.sun.star.text.textfield.TableFormula	Contains a formula to calculate in a text table.
com.sun.star.text.textfield.TemplateName	Shows the name of the template the current document is created from.
com.sun.star.text.textfield.User	Variables - User Field. Creates a global document variable and displays it whenever this field occurs in the text. Depends on com.sun.star.text.FieldMaster.User.
com.sun.star.text.textfield.WordCount	Shows the number of words contained in the document.

All fields support the interfaces com.sun.star.text.XTextField, com.sun.star.util.XUpdatable, com.sun.star.text.XDependentTextField and the service com.sun.star.text.TextContent.

The method getPresentation() of the interface com.sun.star.text.XTextField returns the textual representation of the result of the text field operation, such as a date, time, variable value, or the command, such as CHAPTER, TIME (fixed) depending on the boolean parameter.

The method update() of the interface com.sun.star.util.XUpdatable affects only the following field types:

- Date and time fields are set to the current date and time.
- The ExtendedUser fields that show parts of the user data set for OpenOffice.org, such as the Name, City, Phone No. and the Author fields that are set to the current values.
- The FileName fields are updated with the current name of the file.
- The DocInfo.XXX fields are updated with the current document info of the document.

All other fields ignore calls to update().

Some of these fields need a field master that provides the data that appears in the field. This applies to the field types Database, SetExpression, DDE, User and Bibliography. The interface com.sun.star.text.XDependentTextField handles these pairs of FieldMasters and TextFields. The method attachTextFieldMaster() must be called prior to inserting the field into the document. The method getTextFieldMaster() does not work unless the dependent field is inserted into the document.

To create a valid text field master, the instance has to be created using the `com.sun.star.lang.XMultiServiceFactory` interface of the model with the appropriate service name:

Text Field Master Service Names	Description
`com.sun.star.text.FieldMaster.User`	Contains the global variable that is created and displayed by the fieldtype `com.sun.star.text.textfield.User`.
`com.sun.star.text.FieldMaster.DDE`	The DDE command for a `com.sun.star.text.textfield.DDE`.
`com.sun.star.text.FieldMaster.SetExpression`	Numbering settings if the corresponding `com.sun.star.text.textfield.SetExpression` is a number range. A sub type of expression.
`com.sun.star.text.FieldMaster.Database`	Data source definition for a `com.sun.star.text.textfield.Database`.
`com.sun.star.text.FieldMaster.Bibliography`	Display settings and sorting for `com.sun.star.text.textfield.Bibliography`.

The property `Name` has to be set after the field instance is created, except for the `Database` field master type where the properties `DatabaseName`, `DatabaseTableName`, `DataColumnName` and `DatabaseCommandType` are set instead of the `Name` property.

To access existing text fields and field masters, use the interface `com.sun.star.text.XTextFieldsSupplier` that is implemented at the text document model.

Its method `getTextFields()` returns a `com.sun.star.text.TextFields` container which is a `com.sun.star.container.XEnumerationAccess` and can be refreshed through the `refresh()` method in its interface `com.sun.star.util.XRefreshable`.

Its method `getTextFieldMasters()` returns a `com.sun.star.text.TextFieldMasters` container holding the text field masters of the document. This container provides a `com.sun.star.container.XNameAccess` interface. All field masters, except for `Database` are named by the service name followed by the name of the field master. The `Database` field masters create their names by appending the `DatabaseName`, `DataTableName` and `DataColumnName` to the service name.

Consider the following examples for this naming convention:

"com.sun.star.text.FieldMaster.SetExpression.Illustration"	Master for Illustration number range. Number ranges are built-in `SetExpression` fields present in every document.
"com.sun.star.text.FieldMaster.User.Company"	Master for User field (global document variable), inserted with display name Company.
"com.sun.star.text.FieldMaster.Database.Bibliography.biblio.Identifier"	Master for form letter field referring to the column Identifier in the built-in dbase database table biblio.

Each text field master has a property `InstanceName` that contains its name in the format of the related container.

Some `SetExpression` text field masters are always available if they are not deleted. These are the masters with the names Text, Illustration, Table and Drawing. They are predefined as number

range field masters used for captions of text frames, graphics, text tables and drawings. Note that these predefined names are internal names that are usually not used at the user interface.

The following methods show how to create and insert text fields. (Text/TextDocuments.java)

```
/** This method inserts both a date field and a user field containing the number '42'
 */
protected void TextFieldExample() {
    try {
        // Use the text document's factory to create a DateTime text field,
        // and access it's
        // XTextField interface
        XTextField xDateField = (XTextField) UnoRuntime.queryInterface(
            XTextField.class, mxDocFactory.createInstance(
                "com.sun.star.text.TextField.DateTime"));

        // Insert it at the end of the document
        mxDocText.insertTextContent ( mxDocText.getEnd(), xDateField, false );

        // Use the text document's factory to create a user text field,
        // and access it's XDependentTextField interface
        XDependentTextField xUserField = (XDependentTextField) UnoRuntime.queryInterface (
            XDependentTextField.class, mxDocFactory.createInstance(
                "com.sun.star.text.TextField.User"));

        // Create a fieldmaster for our newly created User Text field, and access it's
        // XPropertySet interface
        XPropertySet xMasterPropSet = (XPropertySet) UnoRuntime.queryInterface(
            XPropertySet.class, mxDocFactory.createInstance(
                "com.sun.star.text.FieldMaster.User"));

        // Set the name and value of the FieldMaster
        xMasterPropSet.setPropertyValue ("Name", "UserEmperor");
        xMasterPropSet.setPropertyValue ("Value", new Integer(42));

        // Attach the field master to the user field
        xUserField.attachTextFieldMaster (xMasterPropSet);

        // Move the cursor to the end of the document
        mxDocCursor.gotoEnd(false);
        // insert a paragraph break using the XSimpleText interface
        mxDocText.insertControlCharacter(
            mxDocCursor, ControlCharacter.PARAGRAPH_BREAK, false);

        // Insert the user field at the end of the document
        mxDocText.insertTextContent(mxDocText.getEnd(), xUserField, false);
    } catch (Exception e) {
        e.printStackTrace (System.out);
    }
}
```

7.3.6 Bookmarks

A Bookmark is a text content that marks a position inside of a paragraph or a text selection that supports the `com.sun.star.text.TextContent` service. To search for a bookmark, the text document model implements the interface `com.sun.star.text.XBookmarksSupplier` that supplies a collection of the bookmarks. The collection supports the service `com.sun.star.text.Bookmarks` which consists of `com.sun.star.container.XNameAccess` and `com.sun.star.container.XIndexAccess`.

The bookmark name can be read and changed through its (`com.sun.star.container.XNamed`) interface.

To insert, remove or change text, or attributes starting from the position of a bookmark, retrieve its `com.sun.star.text.XTextRange` by calling `getAnchor()` at its `com.sun.star.text.XTextContent` interface. Then use `getString()` or `setString()` at the `XTextRange`, or pass this `XTextRange` to methods expecting a text range, such as `com.sun.star.text.XText:createTextCursorByRange()`, `com.sun.star.text.XText:insertString()` or `com.sun.star.text.XText:insertTextContent()`.



Make sure that the access to the bookmark anchor position always uses the correct text object. Since every `XTextRange` knows its surrounding text, use the `getText()` method of the bookmark's anchor. It is not allowed to call `aText.createTextCursorByRange(oAnchor)` when `aText` represents a different area of the document than the bookmark (different text frames, body text and text frame...)

Use the `createInstance` method of the `com.sun.star.lang.XMultiServiceFactory` interface provided by the text document model to insert an new bookmark into the document. The service name is "`com.sun.star.text.Bookmark`". Then use the bookmark's `com.sun.star.container.XNamed` interface and call `setName()`. If no name is set, OpenOffice.org makes up generic names, such as Bookmark1 and Bookmark2. Similarly, if a name is used that is not unique, writer automatically appends a number to the bookmark name. The bookmark object obtained from `createInstance()` can only be inserted once.

```
// inserting and retrieving a bookmark
Object bookmark = mxDocFactory.createInstance ( "com.sun.star.text.Bookmark" );

// name the bookmark
XNamed xNamed = (XNamed) UnoRuntime.queryInterface (
        XNamed.class, bookmark );
xNamed.setName("MyUniqueBookmarkName");

// get XTextContent interface
XTextContent xTextContent = (XTextContent) UnoRuntime.queryInterface (
        XTextContent.class, bookmark );

// insert bookmark at the end of the document
// instead of mxDocText.getEnd you could use a text cursor's XTextRange interface or any XTextRange
mxDocText.insertTextContent ( mxDocText.getEnd(), xTextContent, false );

// query XBookmarksSupplier from document model and get bookmarks collection
XBookmarksSupplier xBookmarksSupplier = (XBookmarksSupplier)UnoRuntime.queryInterface(
        XBookmarksSupplier.class, xWriterComponent);
XNameAccess xNamedBookmarks = xBookmarksSupplier.getBookmarks();

// retrieve bookmark by name
Object foundBookmark = xNamedBookmarks.getByName("MyUniqueBookmarkName");
XTextContent xFoundBookmark = (XTextContent)UnoRuntime.queryInterface(
        XTextContent.class, foundBookmark);

// work with bookmark
XTextRange xFound = xFoundBookmark.getAnchor();
xFound.setString(" The throat mike, glued to her neck, "
        + "looked as much as possible like an analgesic dermadisk.");
```

7.3.7 Indexes and Index Marks

Indexes are text contents that pull together information that is dispersed over the document. They can contain chapter headings, locations of key words, locations of arbitrary index marks and locations of text objects, such as illustrations, objects or tables. In addition, OpenOffice.org features a bibliographical index.

Indexes

The following index services are available in OpenOffice.org:

Index Service Name	Description
`com.sun.star.text.DocumentIndex`	alphabetical index
`com.sun.star.text.ContentIndex`	table of contents
`com.sun.star.text.UserIndex`	user defined index
`com.sun.star.text.IllustrationIndex`	table of all illustrations contained in the document
`com.sun.star.text.ObjectIndex`	table of all objects contained in the document

Index Service Name	Description
com.sun.star.text.TableIndex	table of all text tables contained in the document
com.sun.star.text.Bibliography	bibliographical index

To access the indexes of a document, the text document model supports the interface com.sun.star.text.XDocumentIndexesSupplier with a single method getDocumentIndexes(). The returned object is a com.sun.star.text.DocumentIndexes service supporting the interfaces com.sun.star.container.XIndexAccess and com.sun.star.container.XNameAccess.

All indexes support the services com.sun.star.text.TextContent and com.sun.star.text.BaseIndex that include the interface com.sun.star.text.XDocumentIndex. This interface is used to access the service name of the index and update the current content of an index:

```
string getServiceName()
void update()
```

Furthermore, indexes have properties and a name, and support:

- com.sun.star.beans.XPropertySet
  provides the properties that determine how the index is created and which elements are included into the index.
- com.sun.star.container.XNamed
  provides a unique name of the index, not necessarily the title of the index.

An index is usually composed of two text sections which are provided as properties. The provided property ContentSection includes the complete index and the property HeaderSection contains the title if there is one. They enable the index to have background or column attributes independent of the surrounding page format valid at the index position. In addition, there may be different settings for the content and the heading of the index. However, these text sections are not part of the document's text section container.

The indexes are structured by levels. The number of levels depends on the index type. The content index has ten levels, corresponding to the number of available chapter numbering levels, which is ten. Alphabetical indexes have four levels, one of which is used to insert separators, that are usually characters that show the alphabet. The bibliography has 22 levels, according to the number of available bibliographical type entries (com.sun.star.text.BibliographyDataType). All other index types only have one level.

For all levels, define a separate structure that is provided by the property LevelFormat of the service com.sun.star.text.BaseIndex. LevelFormat contains the various levels as a com.sun.star.container.XIndexReplace object. Each level is a sequence of com.sun.star.beans.PropertyValues which are defined in the service com.sun.star.text.DocumentIndexLevelFormat. Although LevelFormat provides a level for the heading, changing that level is not supported.

Each com.sun.star.beans.PropertyValues sequence has to contain at least one com.sun.star.beans.PropertyValue with the name TokenType. This PropertyValue struct must contain one of the following string values in its Value member variable:

TokenType Value (String)	Meaning	Additional Sequence Members (optional)
"TokenEntryNumber"	The number of an entry. This is only supported in tables of content and it marks the appearance of the chapter number.	CharacterStyleName

TokenType Value (String)	Meaning	Additional Sequence Members (optional)
"TokenEntryText"	Text of the entry, for example, it might contain the heading text in tables of content or the name of a text reference in a bibliography.	CharacterStyleName
"TokenTabStop"	Marks a tab stop to be inserted.	TabStopPosition TabStopRightAligned TabStopFillCharac-ters CharacterStyleName
"TokenText"	Inserted text.	CharacterStyleName Text
"TokenPageNumber"	Marks the insertion of the page number.	CharacterStyleName
"TokenChapterInfo"	Marks the insertion of a chapter field to be inserted. Only supported in alphabetical indexes.	CharacterStyleName ChapterFormat
"TokenHyperlinkStart"	Start of a hyperlink to jump to the referred heading. Only supported in tables of content.	
"TokenHyperlinkEnd"	End of a hyperlink to jump to the referred heading. Only supported in tables of content.	
"TokenBibliographyDa-taField"	Identifies one of the 30 possible BibliographyDataFields. The number 30 comes from the IDL reference of BilbliographyDataFields.	BibliographyDa-taField CharacterStyleName

An example for such a sequence of `PropertyValue` struct could be constructed like this:

```
PropertyValue[] indexTokens = new PropertyValue[1];
indexTokens [0] = new PropertyValue();
indexTokens [0].Name = "TokenType";
indexTokens [0].Value = "TokenHyperlinkStart";
```

The following table explains the sequence members which can be present, in addition to the `TokenType` member, as mentioned above.

Additional Properties of `com.sun.star.text.DocumentIndexLevelFormat`	
CharacterStyleName	`string` — Name of the character style that has to be applied to the appearance of the entry.
TabStopPosition	`long` — Position of the tab stop in 1/100 mm.
TabStopRightAligned	`boolean` — The tab stop is to be inserted at the end of the line and right aligned. This is used before page number entries.
TabStopFillCharacters	`string` — The first character of this string is used as a fill character for the tab stop.
ChapterFormat	`short` — Type of the chapter info as defined in `com.sun.star.text.ChapterFormat`.
BibliographyDataField	Type of the bibliographical entry as defined in `com.sun.star.text.BibliographyDataField`.

Index marks

Index marks are text contents whose contents and positions are collected and displayed in indexes.

To access all index marks that are related to an index, use the property `IndexMarks` of the index. It contains a sequence of `com.sun.star.text.XDocumentIndexMark` interfaces.

All index marks support the service `com.sun.star.text.BaseIndexMark` that includes `com.sun.star.text.TextContent`. Also, they all implement the interfaces `com.sun.star.text.XDocumentIndexMark` and `com.sun.star.beans.XPropertySet`.

The `XDocumentIndexMark` inherits from `XTextContent` and defines two methods:

```
string getMarkEntry()
void setMarkEntry( [in] string anIndexEntry)
```

OpenOffice.org supports three different index mark services:

- `com.sun.star.text.DocumentIndexMark` for entries in alphabetical indexes.
- `com.sun.star.text.UserIndexMark` for user defined indexes.
- `com.sun.star.text.ContentIndexMark` for entries in tables of content which are independent from chapter headings.

An index mark can be set at a point in text or it can mark a portion of a paragraph, usually a word. It cannot contain text across paragraph breaks. If the index mark does not include text, the `BaseIndexMark` property `AlternativeText` has to be set, otherwise there will be no string to insert into the index.

Inserting `ContentIndexMarks` and a table of contents index: (Text/TextDocuments.java)

```
/** This method demonstrates how to insert indexes and index marks
 */
protected void IndexExample ()
{
        try
        {
                // Go to the end of the document
                mxDocCursor.gotoEnd( false );
                // Insert a new paragraph and position the cursor in it
                mxDocText.insertControlCharacter ( mxDocCursor, ControlCharacter.PARAGRAPH_BREAK, false
);
                XParagraphCursor xParaCursor = (XParagraphCursor)
                        UnoRuntime.queryInterface( XParagraphCursor.class, mxDocCursor );
                xParaCursor.gotoPreviousParagraph ( false );

                // Create a new ContentIndexMark and get it's XPropertySet interface
                XPropertySet xEntry = (XPropertySet) UnoRuntime.queryInterface( XPropertySet.class,
                        mxDocFactory.createInstance ( "com.sun.star.text.ContentIndexMark" ) );

                // Set the text to be displayed in the index
                xEntry.setPropertyValue ( "AlternativeText", "Big dogs! Falling on my head!" );

                // The Level property _must_ be set
                xEntry.setPropertyValue ( "Level", new Short ( (short) 1 ) );

                // Create a ContentIndex and access it's XPropertySet interface
                XPropertySet xIndex = (XPropertySet) UnoRuntime.queryInterface( XPropertySet.class,
                        mxDocFactory.createInstance ( "com.sun.star.text.ContentIndex" ) );

                // Again, the Level property _must_ be set
                xIndex.setPropertyValue ( "Level", new Short ( (short) 10 ) );

                // Access the XTextContent interfaces of both the Index and the IndexMark
                XTextContent xIndexContent = (XTextContent) UnoRuntime.queryInterface(
                        XTextContent.class, xIndex );
                XTextContent xEntryContent = (XTextContent) UnoRuntime.queryInterface(
                        XTextContent.class, xEntry );

                // Insert both in the document
                mxDocText.insertTextContent ( mxDocCursor, xEntryContent, false );
                mxDocText.insertTextContent ( mxDocCursor, xIndexContent, false );

                // Get the XDocumentIndex interface of the Index
                XDocumentIndex xDocIndex = (XDocumentIndex) UnoRuntime.queryInterface(
                        XDocumentIndex.class, xIndex );
```

```
                // And call it's update method
                xDocIndex.update();
        }
        catch (Exception e)
        {
                e.printStackTrace ( System.out );
        }

}
```

7.3.8 Reference Marks

A reference mark is a text content that is used as a target for `com.sun.star.text.textfield.GetReference` text fields. These text fields show the contents of reference marks in a text document and allows the user to jump to the reference mark. Reference marks support the `com.sun.star.text.XTextContent` and `com.sun.star.container.XNamed` interfaces. They can be accessed by using the text document's `com.sun.star.text.XReferenceMarksSupplier` interface that defines a single method `getReferenceMarks()`.

The returned collection is a `com.sun.star.text.ReferenceMarks` service which has a `com.sun.star.container.XNameAccess` and a `com.sun.star.container.XIndexAccess` interface. (Text/TextDocuments.java)

```
/** This method demonstrates how to create and insert reference marks, and GetReference Text Fields
 */
protected void ReferenceExample () {
    try {
        // Go to the end of the document
        mxDocCursor.gotoEnd(false);

        // Insert a paragraph break
        mxDocText.insertControlCharacter(
            mxDocCursor, ControlCharacter.PARAGRAPH_BREAK, false);

        // Get the Paragraph cursor
        XParagraphCursor xParaCursor = (XParagraphCursor) UnoRuntime.queryInterface(
            XParagraphCursor.class, mxDocCursor);

        // Move the cursor into the new paragraph
        xParaCursor.gotoPreviousParagraph(false);

        // Create a new ReferenceMark and get it's XNamed interface
        XNamed xRefMark = (XNamed) UnoRuntime.queryInterface(XNamed.class,
            mxDocFactory.createInstance("com.sun.star.text.ReferenceMark"));

        // Set the name to TableHeader
        xRefMark.setName("TableHeader");

        // Get the TextTablesSupplier interface of the document
        XTextTablesSupplier xTableSupplier = (XTextTablesSupplier) UnoRuntime.queryInterface(
            XTextTablesSupplier.class, mxDoc);

        // Get an XIndexAccess of TextTables
        XIndexAccess xTables = (XIndexAccess) UnoRuntime.queryInterface(
            XIndexAccess.class, xTableSupplier.getTextTables());

        // We've only inserted one table, so get the first one from index zero
        XTextTable xTable = (XTextTable) UnoRuntime.queryInterface(
            XTextTable.class, xTables.getByIndex(0));

        // Get the first cell from the table
        XText xTableText = (XText) UnoRuntime.queryInterface(
            XText.class, xTable.getCellByName("A1"));

        // Get a text cursor for the first cell
        XTextCursor xTableCursor = xTableText.createTextCursor();

        // Get the XTextContent interface of the reference mark so we can insert it
        XTextContent xContent = (XTextContent) UnoRuntime.queryInterface(
            XTextContent.class, xRefMark);

        // Insert the reference mark into the first cell of the table
        xTableText.insertTextContent (xTableCursor, xContent, false);
```

```
        // Create a 'GetReference' text field to refer to the reference mark we just inserted,
        // and get it's XPropertySet interface
        XPropertySet xFieldProps = (XPropertySet) UnoRuntime.queryInterface(
            XPropertySet.class, mxDocFactory.createInstance(
                "com.sun.star.text.TextField.GetReference"));

        // Get the XReferenceMarksSupplier interface of the document
        XReferenceMarksSupplier xRefSupplier = (XReferenceMarksSupplier) UnoRuntime.queryInterface(
            XReferenceMarksSupplier.class, mxDoc);

        // Get an XNameAccess which refers to all inserted reference marks
        XNameAccess xMarks = (XNameAccess) UnoRuntime.queryInterface(XNameAccess.class,
            xRefSupplier.getReferenceMarks());

        // Put the names of each reference mark into an array of strings
        String[] aNames = xMarks.getElementNames();

        // Make sure that at least 1 reference mark actually exists
        // (well, we just inserted one!)
        if (aNames.length > 0) {
            // Output the name of the first reference mark ('TableHeader')
            System.out.println ("GetReference text field inserted for ReferenceMark : "
                + aNames[0]);

            // Set the SourceName of the GetReference text field to 'TableHeader'
            xFieldProps.setPropertyValue("SourceName", aNames[0]);

            // specify that the source is a reference mark (could also be a footnote,
            // bookmark or sequence field)
            xFieldProps.setPropertyValue ("ReferenceFieldSource", new Short(
                ReferenceFieldSource.REFERENCE_MARK));

            // We want the reference displayed as 'above' or 'below'
            xFieldProps.setPropertyValue("ReferenceFieldPart",
                new Short (ReferenceFieldPart.UP_DOWN));

            // Get the XTextContent interface of the GetReference text field
            XTextContent xRefContent = (XTextContent) UnoRuntime.queryInterface(
                XTextContent.class, xFieldProps);

            // Go to the end of the document
            mxDocCursor.gotoEnd(false);

            // Make some text to precede the reference
            mxDocText.insertString(mxDocText.getEnd(), "The table ", false);

            // Insert the text field
            mxDocText.insertTextContent(mxDocText.getEnd(), xRefContent, false);

            // And some text after the reference..
            mxDocText.insertString( mxDocText.getEnd(),
                " contains the sum of some random numbers.", false);

            // Refresh the document
            XRefreshable xRefresh = (XRefreshable) UnoRuntime.queryInterface(
                XRefreshable.class, mxDoc);
            xRefresh.refresh();
        }
    } catch (Exception e) {
        e.printStackTrace(System.out);
    }
}
```

The name of a reference mark can be used in a `com.sun.star.text.textfield.GetReference` text field to refer to the position of the reference mark.

7.3.9 Footnotes and Endnotes

Footnotes and endnotes are text contents that provide background information for the reader that appears in page footers or at the end of a document.

Footnotes and endnotes implement the service `com.sun.star.text.Footnote` that includes `com.sun.star.text.TextContent`. The `Footnote` service has the interfaces `com.sun.star.text.XText` and `com.sun.star.text.XFootnote` that inherit from `com.sun.star.text.XTextContent`. The `XFootnote` introduces the following methods:

```
string getLabel()
void setLabel( [in] string aLabel)
```

The `Footnote` service defines a property `ReferenceId` that is used for import and export, and contains an internal sequential number.

The interface `com.sun.star.text.XText` which is provided by the `com.sun.star.text.Footnote` service accesses the text object in the footnote area where the footnote text is located. It is not allowed to insert text tables into this text object.

While footnotes can be placed at the end of a page or the end of a document, endnotes always appear at the end of a document. Endnote numbering is separate from footnote numbering. Footnotes are accessed using the `com.sun.star.text.XFootnotesSupplier` interface of the text document through the method `getFootNotes()`. Endnotes are accessed similarly by calling `getEndnotes()` at the text document's `com.sun.star.text.XEndnotesSupplier` interface. Both of these methods return a `com.sun.star.container.XIndexAccess`.

A label is set for a footnote or endnote to determine if automatic footnote numbering is used. If no label is set (= empty string), the footnote is labeled automatically. There are footnote and endnote settings that specify how the automatic labeling is formatted. These settings are obtained from the document model using the interfaces `com.sun.star.text.XFootnotesSupplier` and `com.sun.star.text.XEndnotesSupplier`. The corresponding methods are `getFootnoteSettings()` and `getEndnoteSettings()`. The object received is a `com.sun.star.beans.XPropertySet` and has the properties described in `com.sun.star.text.FootnoteSettings`:

Properties of `com.sun.star.text.FootnoteSettings`	
`AnchorCharStyleName`	`string` — Contains the name of the character style that is used for the label in the document text.
`CharStyleName`	`string` — Contains the name of the character style that is used for the label in front of the footnote/endnote text.
`NumberingType`	`short` — Contains the numbering type for the numbering of the footnotes or endnotes.
`PageStyleName`	`string` — Contains the page style that is used for the page that contains the footnote or endnote texts.
`ParaStyleName`	`string` —Contains the paragraph style that is used for the footnote or endnote text.
`Prefix`	`string` —Contains the prefix for the footnote or endnote symbol.
`StartAt`	`short` — Contains the first number of the automatic numbering of footnotes or endnotes.
`Suffix`	`string` — Contains the suffix for the footnote/endnote symbol.
`BeginNotice`	`[optional] string` —Contains the string at the restart of the footnote text after a break.
`EndNotice`	`[optional] string` — Contains the string at the end of a footnote part in front of a break.
`FootnoteCounting`	`[optional] boolean` —Contains the type of the counting for the footnote numbers
`PositionEndOfDoc`	`[optional] boolean` — If `true`, the footnote text is shown at the end of the document.

The `Footnotes` service applies to footnotes and endnotes.

The following sample works with footnotes (Text/TextDocuments.java)

```
/** This method demonstrates how to create and insert footnotes, and how to access the
    XFootnotesSupplier interface of the document
 */
```

```
protected void FootnoteExample ()
{
        try
        {
                // Create a new footnote from the document factory and get it's
                // XFootnote interface
                XFootnote xFootnote = (XFootnote) UnoRuntime.queryInterface( XFootnote.class,
                                mxDocFactory.createInstance ( "com.sun.star.text.Footnote" ) );

                // Set the label to 'Numbers'
                xFootnote.setLabel ( "Numbers" );

                // Get the footnotes XTextContent interface so we can...
                XTextContent xContent = ( XTextContent ) UnoRuntime.queryInterface (
                        XTextContent.class, xFootnote );

                // ...insert it into the document
                mxDocText.insertTextContent ( mxDocCursor, xContent, false );

                // Get the XFootnotesSupplier interface of the document
                XFootnotesSupplier xFootnoteSupplier = (XFootnotesSupplier) UnoRuntime.queryInterface(
                        XFootnotesSupplier.class, mxDoc );

                // Get an XIndexAccess interface to all footnotes
                XIndexAccess xFootnotes = ( XIndexAccess ) UnoRuntime.queryInterface (
                        XIndexAccess.class, xFootnoteSupplier.getFootnotes() );

                // Get the XFootnote interface to the first footnote inserted ('Numbers')
                XFootnote xNumbers = ( XFootnote ) UnoRuntime.queryInterface (
                        XFootnote.class, xFootnotes.getByIndex( 0 ) );

                // Get the XSimpleText interface to the Footnote
                XSimpleText xSimple = (XSimpleText ) UnoRuntime.queryInterface (
                        XSimpleText.class, xNumbers );

                // Create a text cursor for the foot note text
                XTextRange xRange = (XTextRange ) UnoRuntime.queryInterface (
                        XTextRange.class, xSimple.createTextCursor() );

                // And insert the actual text of the footnote.
                xSimple.insertString (
                        xRange, "  The numbers were generated by using java.util.Random", false );
        }
        catch (Exception e)
        {
                e.printStackTrace ( System.out );
        }
}
```

7.3.10 Shape Objects in Text

Base Frames vs. Drawing Shapes

Shape objects are text contents that act independently of the ordinary text flow. The surrounding text may wrap around them. Shape objects can lie in front or behind text, and be anchored to paragraphs or characters in the text. Anchoring allows the shape objects to follow the paragraphs and characters while the user is writing. Currently, there are two different kinds of shape objects in OpenOffice.org, base frames and drawing shapes.

Base Frames

The first group are shape objects that are `com.sun.star.text.BaseFrames`. The three services `com.sun.star.text.TextFrame`, `com.sun.star.text.TextGraphicObject` and `com.sun.star.text.TextEmbeddedObject` are all based on the service `com.sun.star.text.BaseFrame`. The `TextFrames` contain an independent text area that can be positioned freely over ordinary text. The `TextGraphicObjects` are bitmaps or vector oriented images in a format supported by OpenOffice.org internally. The `TextEmbeddedObjects` are areas containing a document type other than the document they are embedded in, such as charts, formulas, internal OpenOffice.org documents (Calc/Draw/Impress), or OLE objects.

The TextFrames, TextGraphicObjects and TextEmbeddedObjects in a text are supplied by their corresponding supplier interfaces at the document model:
com.sun.star.text.XTextFramesSupplier
com.sun.star.text.XTextGraphicObjectsSupplier
com.sun.star.text.XTextEmbeddedObjectsSupplier. These interfaces all have one single get method that supplies the respective Shape objects collection:

```
com::sun::star::container::XNameAccess getTextFrames()
com::sun::star::container::XNameAccess getTextEmbeddedObjects()
com::sun::star::container::XNameAccess getTextGraphicObjects()
```

The method getTextFrames() returns a com.sun.star.text.TextFrames collection, getTextEmbeddedObjects() returns a com.sun.star.text.TextEmbeddedObjects collection and getTextGraphicObjects() yields a com.sun.star.text.TextGraphicObjects collection. All of these collections support com.sun.star.container.XIndexAccess and com.sun.star.container.XNameAccess. The TextFrames collection may (optional) support the com.sun.star.container.XContainer interface to broadcast an event when an Element is added to the collection. However, the current implementation of the TextFrames collection does not support this.

The service com.sun.star.text.BaseFrame defines the common properties and interfaces of text frames, graphic objects and embedded objects. It includes the services com.sun.star.text.BaseFrameProperties and com.sun.star.text.TextContent, and defines the following interfaces.

The position and size of a BaseFrame is covered by com.sun.star.drawing.XShape. All BaseFrame objects share a majority of the core implementation of drawing objects. Therefore, they have a position and size on the DrawPage.

The name of a BaseFrame is set and read through com.sun.star.container.XNamed. The names of the frame objects have to be unique for text frames, graphic objects and embedded objects, respectively.

The com.sun.star.beans.XPropertySet has to be present, because any aspects of BaseFrames are controlled through properties.

The interface com.sun.star.document.XEventsSupplier is not a part of the BaseFrame service, but is available in text frames, graphic objects and embedded objects. This interface provides access to the event macros that may be attached to the object in the GUI.

The properties of BaseFrames are those of the service com.sun.star.text.TextContent, as well there is a number of frame properties defined in the service com.sun.star.text.BaseFrameProperties:

Properties of com.sun.star.text.BaseFrameProperties	
AnchorPageNo	short — Contains the number of the page where the objects are anchored.
AnchorFrame	com.sun.star.text.XTextFrame. Contains the text frame the current frame is anchored to.
BackColor	long — Contains the color of the background of the object.
BackGraphicURL	string — Contains the URL for the background graphic.
BackGraphicFilter	string — Contains the name of the file filter for the background graphic.
BackGraphicLocation	Determines the position of the background graphic according to com.sun.star.style.GraphicLocation.
LeftBorder	struct com.sun.star.table.BorderLine. Contains the left border of the object.

Properties of com.sun.star.text.BaseFrameProperties	
RightBorder	struct com.sun.star.table.BorderLine. Contains the right border of the object.
TopBorder	struct com.sun.star.table.BorderLine. Contains the top border of the object.
BottomBorder	struct com.sun.star.table.BorderLine. Contains the bottom border of the object.
BorderDistance	long — Contains the distance from the border to the object.
LeftBorderDistance	long — Contains the distance from the left border to the object.
RightBorderDistance	long — Contains the distance from the right border to the object.
TopBorderDistance	long — Contains the distance from the top border to the object.
BottomBorderDistance	long — Contains the distance from the bottom border to the object.
BackTransparent	boolean — If true, the property BackColor is ignored.
ContentProtected	boolean — Determines if the content is protected.
LeftMargin	long — Contains the left margin of the object.
RightMargin	long — Contains the right margin of the object.
TopMargin	long — Contains the top margin of the object.
BottomMargin	long — Contains the bottom margin of the object.
Height	long — Contains the height of the object (1/100 mm).
Width	long — Contains the width of the object (1/100 mm).
RelativeHeight	short — Contains the relative height of the object.
RelativeWidth	short — Contains the relative width of the object.
IsSyncWidthToHeight	boolean — Determines if the width follows the height.
IsSyncHeightToWidth	boolean — Determines if the height follows the width.
HoriOrient	short — Determines the horizontal orientation of the object according to com.sun.star.text.HoriOrientation.
HoriOrientPosition	long — Contains the horizontal position of the object (1/100 mm).
HoriOrientRelation	short — Determines the environment of the object the orientation is related according to com.sun.star.text.RelOrientation.
VertOrient	short — Determines the vertical orientation of the object.
VertOrientPosition	long — Contains the vertical position of the object (1/100 mm). Valid only if TextEmbeddedObject::VertOrient is VertOrientation::NONE .
VertOrientRelation	short — Determines the environment of the object the orientation is related according to com.sun.star.text.RelOrientation.
HyperLinkURL	string — Contains the URL of a hyperlink that is set at the object.
HyperLinkTarget	string — Contains the name of the target for a hyperlink that is set at the object.
HyperLinkName	string — Contains the name of the hyperlink that is set at the object.
Opaque	boolean — Determines if the object is opaque or transparent for text.
PageToggle	boolean — Determines if the object is mirrored on even pages.
PositionProtected	boolean — Determines if the position is protected.

Properties of com.sun.star.text.BaseFrameProperties	
Print	boolean — Determines if the object is included in printing.
ShadowFormat	struct com.sun.star.table.ShadowFormat. Contains the type of the shadow of the object.
ServerMap	boolean — Determines if the object gets an image map from a server.
Size	struct com.sun.star.awt.Size. Contains the size of the object.
SizeProtected	boolean — Determines if the size is protected.
Surround	[deprecated]. Determines the type of the surrounding text.
SurroundAnchorOnly	boolean — Determines if the text of the paragraph where the object is anchored, wraps around the object.

Drawing Shapes

The second group of shape objects are the varied drawing shapes that can be inserted into text, such as rectangles and ellipses. They are based on com.sun.star.text.Shape. The service text.Shape includes com.sun.star.drawing.Shape, but adds a number of properties related to shapes in text (cf. *7.3.10 Text Documents - Working with Text Documents - Shape Objects in Text - Drawing Shapes* below). In addition, drawing shapes support the interface com.sun.star.text.XTextContent so that they can be inserted into an XText.

There are no specialized supplier interfaces for drawing shapes. All the drawing shapes on the DrawPage object are supplied by the document model's com.sun.star.drawing.XDrawPageSupplier and its single method:

```
com::sun::star::drawing::XDrawPage getDrawPage()
```



The DrawPage not only contains drawing shapes, but the BaseFrame shape objects too, if the document contains any.

Text Frames

A text frame is a com.sun.star.text.TextFrame service consisting of com.sun.star.text.BaseFrame and the interface com.sun.star.text.XTextFrame.The XText-Frame is based on com.sun.star.text.XTextContent and introduces one method to provide the XText of the frame:

```
com::sun::star::text::XText getText()
```

The properties of com.sun.star.text.TextFrame that add to the BaseFrame are the following:

Properties of com.sun.star.text.TextFrame	
FrameHeightAbsolute	long — Contains the metric height value of the frame.
FrameWidthAbsolute	long — Contains the metric width value of the frame.
FrameWidthPercent	byte — Specifies a width relative to the width of the surrounding text.
FrameHeightPercent	byte — Specifies a width relative to the width of the surrounding text.
FrameIsAutomaticHeight	boolean — If "AutomaticHeight" is set, the object grows if it is required by the frame content.
SizeType	short — Determines the interpretation of the height and relative height properties.

Additionally, text frames are `com.sun.star.text.Text` services and support all of its interfaces, except for `com.sun.star.text.XTextRangeMover`.

Text frames can be connected to a chain, that is, the text of the first text frame flows into the next chain element if it does not fit. The properties `ChainPrevName` and `ChainNextName` are provided to take advantage of this feature. They contain the names of the predecessor and successor of a frame. All frames have to be empty to chain frames, except for the first member of the chain.

Chained Text Frame Property	
`ChainPrevName`	`string` — Name of the predecessor of the frame.
`ChainNextName`	`string` — Name of the successor of the frame.

The effect at the API is that the visible text content of the chain members is only accessible at the first frame in the chain. The content of the following chain members is not shown when chained before their content is set.

The API reference does not know the properties above. Instead, it specifies a `com.sun.star.text.ChainedTextFrame` with an `XChainable` interface, but this is not yet supported by text frames.

The following example uses text frames: (Text/TextDocuments.java)

```
/** This method shows how to create and manipulate text frames
 */
protected void TextFrameExample ()
{
        try
        {
                // Use the document's factory to create a new text frame and immediately access
                // it's XTextFrame interface
                XTextFrame xFrame = (XTextFrame) UnoRuntime.queryInterface (
                        XTextFrame.class, mxDocFactory.createInstance (
                                "com.sun.star.text.TextFrame" ) );

                // Access the XShape interface of the TextFrame
                XShape xShape = (XShape) UnoRuntime.queryInterface(XShape.class, xFrame);
                // Access the XPropertySet interface of the TextFrame
                XPropertySet xFrameProps = (XPropertySet)UnoRuntime.queryInterface(
                        XPropertySet.class, xFrame );

                // Set the size of the new Text Frame using the XShape's 'setSize' method
                Size aSize = new Size();
                aSize.Height = 400;
                aSize.Width = 15000;
                xShape.setSize(aSize);
                // Set the AnchorType to com.sun.star.text.TextContentAnchorType.AS_CHARACTER
                xFrameProps.setPropertyValue( "AnchorType", TextContentAnchorType.AS_CHARACTER );
                // Go to the end of the text document
                mxDocCursor.gotoEnd( false );
                // Insert a new paragraph
                mxDocText.insertControlCharacter (
                        mxDocCursor, ControlCharacter.PARAGRAPH_BREAK, false );
                // Then insert the new frame
                mxDocText.insertTextContent(mxDocCursor, xFrame, false);

                // Access the XText interface of the text contained within the frame
                XText xFrameText = xFrame.getText();
                // Create a TextCursor over the frame's contents
                XTextCursor xFrameCursor = xFrameText.createTextCursor();
                // Insert some text into the frame
                xFrameText.insertString(
                        xFrameCursor, "The first line in the newly created text frame.", false );
                xFrameText.insertString(
                        xFrameCursor, "\nThe second line in the new text frame.", false );
                // Insert a paragraph break into the document (not the frame)
                mxDocText.insertControlCharacter (
                        mxDocCursor, ControlCharacter.PARAGRAPH_BREAK, false );
         }
        catch (Exception e)
        {
                e.printStackTrace ( System.out );
        }
}
```

Embedded Objects

A `TextEmbeddedObject` is a `com.sun.star.text.BaseFrame` providing the interface `com.sun.star.document.XEmbeddedObjectSupplier`. The only method of this interface,

```
com::sun::star::lang::XComponent getEmbeddedObject ()
```

provides access to the model of the embedded document. That way, an embedded OpenOffice.org spreadsheet, drawing, chart or a formula document can be used in a text over its document model.

At this time, there is no support to create and insert embedded objects at the API.

Graphic Objects

A `TextGraphicObject` is a `BaseFrame` and does not provide any additional interfaces, compared with `com.sun.star.text.BaseFrame`. However, it introduces a number of properties that allow manipulating of a graphic object. They are described in the service `com.sun.star.text.TextGraphicObject`:

Properties of `com.sun.star.text.TextGraphicObject`	
`ImageMap`	`com.sun.star.container.XIndexContainer`. Returns the client-side image map if one is assigned to the object.
`ContentProtected`	`boolean` — Determines if the content is protected against changes from the user interface.
`SurroundContour`	`boolean` — Determines if the text wraps around the contour of the object.
`ContourOutside`	`boolean` — The text flows only around the contour of the object.
`ContourPolyPolygon`	`[optional]` struct `com.sun.star.drawing.PointSequenceSequence`. Contains the contour of the object as PolyPolygon.
`GraphicCrop`	struct `com.sun.star.text.GraphicCrop`. Contains the cropping of the object.
`HoriMirroredOnEvenPages`	`boolean` — Determines if the object is horizontally mirrored on even pages.
`HoriMirroredOnOddPages`	`boolean` — Determines if the object is horizontally mirrored on odd pages.
`VertMirrored`	`boolean` — Determines if the object is mirrored vertically.
`GraphicURL`	`string` — Contains the URL of the background graphic of the object.
`GraphicFilter`	`string` — Contains the name of the filter of the background graphic of the object.
`ActualSize`	`com.sun.star.awt.Size`. Contains the original size of the bitmap in the graphic object.
`AdjustLuminance`	`short` — Changes the display of the luminance. It contains percentage values between -100 and +100.
`AdjustContrast`	`short` — Changes the display of contrast. It contains percentage values between -100 and +100.
`AdjustRed`	`short` — Changes the display of the red color channel. It contains percentage values between -100 and +100.

Properties of com.sun.star.text.TextGraphicObject	
AdjustGreen	short — Changes the display of the green color channel. It contains percentage values between -100 and +100.
AdjustBlue	short — Changes the display of the blue color channel. It contains percentage values between -100 and +100.
Gamma	double — Determines the gamma value of the graphic.
GraphicIsInverted	boolean — Determines if the graphic is displayed in inverted colors. It contains percentage values between -100 and +100.
Transparency	short — Measure of transparency. It contains percentage values between -100 and +100.
GraphicColorMode	long — Contains the ColorMode according to com.sun.star.drawing.ColorMode.



TextGraphicObject files can currently only be linked when inserted through API which means only their URL is stored with the document. Embedding of graphics is not supported. This applies to background graphics which can be set, for example, to paragraphs, tables or text sections.

Drawing Shapes

The writer uses the same drawing engine as OpenOffice.org impress and OpenOffice.org draw. The limitations are that in writer only one draw page can exist and 3D objects are not supported. All drawing shapes support these properties:

Properties of com.sun.star.drawing.Shape	
ZOrder	[optional] long — Is used to query or change the ZOrder of this Shape .
LayerID	[optional] short — This is the ID of the layer to which this shape is attached.
LayerName	[optional] string — This is the name of the layer to which this Shape is attached.
Printable	[optional] boolean — If this is false, the shape is not visible on printer outputs.
MoveProtect	[optional] boolean — When set to true, this shape cannot be moved interactively in the user interface.
Name	[optional] string — This is the name of this shape.
SizeProtect	[optional] boolean — When set to true, this shape may not be sized interactively in the user interface.
Style	[optional] com.sun.star.style.XStyle. Determines the style for this shape.
Transformation	[optional] com.sun.star.drawing.HomogenMatrix This property lets you get and set the transformation matrix for this shape. The transformation is a 3x3 blended matrix and can contain translation, rotation, shearing and scaling.
ShapeUserDefinedAttributes	[optional] com.sun.star.container.XNameContainer. This property stores xml attributes. They are saved to and restored from automatic styles inside xml files.

In addition to the properties of the shapes natively supported by the drawing engine, the writer shape adds some properties, so that they are usable for text documents. These are defined in the service `com.sun.star.text.Shape`:

Properties of `com.sun.star.text.Shape`	
`AnchorPageNo`	`short` — Contains the number of the page where the objects are anchored.
`AnchorFrame`	`com.sun.star.text.XTextFrame`. Contains the text frame the current frame is anchored to.
`SurroundAnchorOnly`	`boolean` — Determines if the text of the paragraph in which the object is anchored, wraps around the object.
`AnchorType`	`[optional] com.sun.star.text.TextContentAnchorType`. Specifies how the text content is attached to its surrounding text.
`HoriOrient`	`short` — Determines the horizontal orientation of the object.
`HoriOrientPosition`	`long` — Contains the horizontal position of the object (1/100 mm).
`HoriOrientRelation`	`short` — Determines the environment of the object to which the orientation is related.
`VertOrient`	`short` — Determines the vertical orientation of the object.
`VertOrientPosition`	`long` — Contains the vertical position of the object (1/100 mm). Valid only if TextEmbeddedObject::VertOrient is VertOrientation::NONE.
`VertOrientRelation`	`short` — Determines the environment of the object to which the orientation is related.
`LeftMargin`	`long` — Contains the left margin of the object.
`RightMargin`	`long` — Contains the right margin of the object.
`TopMargin`	`long` — Contains the top margin of the object.
`BottomMargin`	`long` — Contains the bottom margin of the object.
`Surround`	[deprecated]. Determines the type of the surrounding text.
`SurroundAnchorOnly`	`boolean` — Determines if the text of the paragraph in which the object is anchored, wraps around the object.
`SurroundContour`	`boolean` — Determines if the text wraps around the contour of the object.
`ContourOutside`	`boolean` — The text flows only around the contour of the object.
`Opaque`	`boolean` — Determines if the object is opaque or transparent for text.
`TextRange`	`com.sun.star.text.XTextRange`. Contains a text range where the shape should be anchored to.

The chapter *9 Drawing* describes how to use shapes and the interface of the draw page.

A sample that creates and inserts drawing shapes: (Text/TextDocuments.java)

```
/** This method demonstrates how to create and manipulate shapes, and how to access the draw page
    of the document to insert shapes
 */
protected void DrawPageExample () {
    try {
        // Go to the end of the document
        mxDocCursor.gotoEnd(false);
        // Insert two new paragraphs
        mxDocText.insertControlCharacter(mxDocCursor,
        ControlCharacter.PARAGRAPH_BREAK, false);
        mxDocText.insertControlCharacter(mxDocCursor,
            ControlCharacter.PARAGRAPH_BREAK, false);

        // Get the XParagraphCursor interface of our document cursor
        XParagraphCursor xParaCursor = (XParagraphCursor)
            UnoRuntime.queryInterface(XParagraphCursor.class, mxDocCursor);
```

```
        // Position the cursor in the 2nd paragraph
        xParaCursor.gotoPreviousParagraph(false);

        // Create a RectangleShape using the document factory
        XShape xRect = (XShape) UnoRuntime.queryInterface(
            XShape.class, mxDocFactory.createInstance(
                "com.sun.star.drawing.RectangleShape"));

        // Create an EllipseShape using the document factory
        XShape xEllipse = (XShape) UnoRuntime.queryInterface(
            XShape.class, mxDocFactory.createInstance(
                "com.sun.star.drawing.EllipseShape"));

        // Set the size of both the ellipse and the rectangle
        Size aSize = new Size();
        aSize.Height = 4000;
        aSize.Width = 10000;
        xRect.setSize(aSize);
        aSize.Height = 3000;
        aSize.Width = 6000;
        xEllipse.setSize(aSize);

        // Set the position of the Rectangle to the right of the ellipse
        Point aPoint = new Point();
        aPoint.X = 6100;
        aPoint.Y = 0;
        xRect.setPosition (aPoint);

        // Get the XPropertySet interfaces of both shapes
        XPropertySet xRectProps = (XPropertySet) UnoRuntime.queryInterface(
            XPropertySet.class, xRect);
        XPropertySet xEllipseProps = (XPropertySet) UnoRuntime.queryInterface(
            XPropertySet.class, xEllipse);

        // And set the AnchorTypes of both shapes to 'AT_PARAGRAPH'
        xRectProps.setPropertyValue("AnchorType", TextContentAnchorType.AT_PARAGRAPH);
        xEllipseProps.setPropertyValue("AnchorType", TextContentAnchorType.AT_PARAGRAPH);

        // Access the XDrawPageSupplier interface of the document
        XDrawPageSupplier xDrawPageSupplier = (XDrawPageSupplier) UnoRuntime.queryInterface(
             XDrawPageSupplier.class, mxDoc);

        // Get the XShapes interface of the draw page
        XShapes xShapes = (XShapes) UnoRuntime.queryInterface(
            XShapes.class, xDrawPageSupplier.getDrawPage());

        // Add both shapes
        xShapes.add (xEllipse);
        xShapes.add (xRect);

        /*
        This doesn't work, I am assured that FME and AMA are fixing it.

        XShapes xGrouper = (XShapes) UnoRuntime.queryInterface(
            XShapes.class, mxDocFactory.createInstance(
                "com.sun.star.drawing.GroupShape"));

        XShape xGrouperShape = (XShape) UnoRuntime.queryInterface(XShape.class, xGrouper);
        xShapes.add (xGrouperShape);

        xGrouper.add (xRect);
        xGrouper.add (xEllipse);

        XShapeGrouper xShapeGrouper = (XShapeGrouper) UnoRuntime.queryInterface(
            XShapeGrouper.class, xShapes);
        xShapeGrouper.group (xGrouper);
        */

    } catch (Exception e) {
        e.printStackTrace(System.out);
    }
}
```

7.3.11 Redline

Redlines are text portions created in the user interface by switching on **Edit - Changes - Record**. Redlines in a document are accessed through the `com.sun.star.document.XRedlinesSupplier` interface at the document model. A collection of redlines as `com.sun.star.beans.XPropertySet` objects are received that can be accessed as `com.sun.star.container.XIndexAccess` or as

`com.sun.star.container.XEnumerationAccess`. Their properties are described in `com.sun.star.text.RedlinePortion`.

If a change is recorded, but not visible because the option **Edit - Changes - Show** has been switched off, redline text is contained in the property `RedlineText`, which is a `com.sun.star.text.XText`.

Calling `XPropertySet.getPropertySetInfo()` on a redline property set crashes the office.

7.3.12 Ruby

Ruby text is a character layout attribute used in Asian languages. Ruby text appears above or below text in left to right writing, and left to right of text in top to bottom writing. For examples, cf. *www.w3.org/TR/1999/WD-ruby-19990322/*.

Ruby text is created using the appropriate character properties from the service `com.sun.star.style.CharacterProperties` wherever this service is supported. However, theAsian languages support must be switched on in **Tools - Options - LanguageSettings - Languages**.

There is no convenient supplier interface for ruby text at the model at this time. However, the controller has an interface `com.sun.star.text.XRubySelection` that provides access to rubies contained in the current selection.

To find ruby text in the model, enumerate all text portions in all paragraphs and check if the property `TextPortionType` contains the string `"Ruby"` to find ruby text. When there is ruby text, access the `RubyText` property of the text portion that contains ruby text as a string.

CharacterProperties for Ruby Text	Description
`com.sun.star.style.CharacterProperties:RubyText`	Contains the text that is set as ruby.
`com.sun.star.style.CharacterProperties:RubyAdjust`	Determines the adjustment of the ruby text as `RubyAdjust`.
`com.sun.star.style.CharacterProperties:RubyCharStyleName`	Contains the name of the character style that is applied to `RubyText`.
`com.sun.star.style.CharacterProperties:RubyIsAbove`	Determines if the ruby text is printed above/left or below/right of the text

7.4 Overall Document Features

7.4.1 Styles

Styles distinguish sections in a document that are commonly formatted and separates this information from the actual formatting. This way it is possible to unify the appearance of a document, and adjust the formatting of a document by altering a style, instead of local format settings after the document has been completed. Styles are packages of attributes that can be applied to text or text contents in a single step.

The following style families are available in OpenOffice.org.

Style Families	Description
CharacterStyles	Character styles are used to format single characters or entire words and phrases. Character styles can be nested.
ParagraphStyles	Paragraph styles are used to format entire paragraphs. Apart from the normal format settings for paragraphs, the paragraph style also defines the font to be used, and the paragraph style for the following paragraph.
FrameStyles	Frame styles are used to format graphic and text frames. These Styles are used to quickly format graphics and frames automatically.
PageStyles	Page styles are used to structure the page. If a "Next Style" is specified, the OpenOffice.org automatically applies the specified page style when an automatic page break occurs.
NumberingStyles	Numbering styles are used to format paragraphs in numbered or bulleted text.

The text document model implements the interface `com.sun.star.style.XStyleFamiliesSupplier` to access these styles. Its method `getStyleFamilies()` returns a collection of `com.sun.star.style.StyleFamilies` with a `com.sun.star.container.XNameAccess` interface. The `com.sun.star.container.XNameAccess` interface retrieves the style families by the names listed above. The `StyleFamilies` service supports a `com.sun.star.container.XIndexAccess`.

From the `StyleFamilies`, retrieve one of the families listed above by name or index. A collection of styles are received which is a `com.sun.star.style.StyleFamily` service, providing access to the single styles through an `com.sun.star.container.XNameContainer` or an `com.sun.star.container.XIndexAccess`.

Each style is a `com.sun.star.style.Style` and supports the interface `com.sun.star.style.XStyle` that inherits from `com.sun.star.container.XNamed`. The `XStyle` contains:

```
string getName()
void setName( [in] string aName)
boolean isUserDefined()
boolean isInUse()
string getParentStyle()
void setParentStyle( [in] string aParentStyle)
```

The office comes with a set of default styles. These styles use programmatic names on the API level. The method `setName()` in `XStyle` always throws an exception if called at such styles. The same applies to changing the property `Category`. At the user interface localized names are used. The user interface names are provided through the property `UserInterfaceName`.

Note that page and numbering styles are not hierarchical and cannot have parent styles. The method `getParentStyle()` always returns an empty string, and the method `setParentStyle()` throws a `com.sun.star.uno.RuntimeException` when called at a default style.

The method `isUserDefined()` determines whether a style is defined by a user or is a built-in style. A built-in style cannot be deleted. Additionally the built-in styles have two different names: a true object name and an alias that is displayed at the user interface. This is not usually visible in an English OpenOffice.org version, except for the default styles that are named "Standard" as programmatic name and "Default" in the user interface.

The Style service defines the following properties which are shared by all styles:

Properties of `com.sun.star.style.Style`	
`IsPhysical`	`[optional, readonly] boolean` — Determines if a style is physically created.
`FollowStyle`	`[optional] boolean` — Contains the name of the style that is applied to the next paragraph.

Properties of com.sun.star.style.Style	
DisplayName	[optional, readonly] string — Contains the name of the style as is displayed in the user interface.
IsAutoUpdate	[optional] string — Determines if a style is automatically updated when the properties of an object that the style is applied to are changed.

To determine the user interface name, each style has a string property DisplayName that contains the name that is used at the user interface. It is not allowed to use a DisplayName of a style as a name of a user-defined style of the same style family.

The built-in styles are not created actually as long as they are not used in the document. The property IsPhysical checks for this. It is necessary, for file export purposes, to detect styles which do not need to be exported.

The StyleFamilies collection can load styles. For this purpose, the interface com.sun.star.style.XStyleLoader is available at the StyleFamilies collection. It consists of two methods:

```
void loadStylesFromURL( [in] string URL,
                        [in] sequence< com::sun::star::beans::PropertyValue > aOptions)
sequence< com::sun::star::beans::PropertyValue > getStyleLoaderOptions()
```

The method loadStylesFromURL() enables the document to import styles from other documents. The expected sequence of PropertyValue structs can contain the following properties:

Properties for loadStylesFromURL()	Description
LoadTextStyles	Determines if character and paragraph styles are to be imported. It is not possible to select character styles and paragraph styles separately.
LoadLoadFrameStyles	boolean — Import frame styles only.
LoadPageStyles	boolean — Import page styles only.
LoadNumberingStyles	boolean — Import numbering styles only.
OverwriteStyles	boolean — Determines if internal styles are overwritten if the source document contains styles having the same name.

The method getStyleLoaderOptions() returns a sequence of these PropertyValue structs, set to their default values.

Character Styles

Character styles support all properties defined in the services com.sun.star.style.CharacterProperties and com.sun.star.style.CharacterPropertiesAsian.

They are created using the com.sun.star.lang.XMultiServiceFactory interface of the text document model using the service name "com.sun.star.style.CharacterStyle".

The default style that is shown in the user interface and accessible through the API is not a style, but a tool to remove applied character styles. Therefore, its properties cannot be changed.

Set the property CharStyleName at an object including the service com.sun.star.style.CharacterProperties to set its character style.

Paragraph Styles

Paragraph styles support all properties defined in the services `com.sun.star.style.ParagraphProperties` and `com.sun.star.style.ParagraphPropertiesAsian`.

They are created using the `com.sun.star.lang.XMultiServiceFactory` interface of the text document model using the service name "`com.sun.star.style.ParagraphStyle`".

Additionally, there is a service `com.sun.star.style.ConditionalParagraphStyle` which creates conditional paragraph styles. Conditional styles are paragraph styles that have different effects, depending on the context. There is currently no support of the condition properties at the API.

Set the property `ParaStyleName` at an object, including the service `com.sun.star.style.ParagraphProperties` to set its paragraph style.

Frame Styles

Frame styles support all properties defined in the services `com.sun.star.text.BaseFrameProperties`.

The frame styles are applied to text frames, graphic objects and embedded objects.

They are created using the `com.sun.star.lang.XMultiServiceFactory` interface of the text document model using the service name "`com.sun.star.style.FrameStyle`".

Set the property `FrameStyleName` at `com.sun.star.text.BaseFrame` objects to set their frame style.

Page Styles

Page styles are controlled via properties. The page related properties are defined in the services `com.sun.star.style.PageStyle`

They are created using the `com.sun.star.lang.XMultiServiceFactory` interface of the text document model using the service name "`com.sun.star.style.PageStyle`".

As mentioned above, page styles are not hierarchical. The section *7.4.5 Text Documents - Overall Document Features - Page Layout* discusses page styles.

The `PageStyle` is set at the current text cursor position. Set the property `PageStyleName` to change the page style, and use the property `PageDescName` to insert a new page, changing the page style.

Numbering Styles

Numbering styles support all properties defined in the services `com.sun.star.text.NumberingStyle`.

They are created using the `com.sun.star.lang.XMultiServiceFactory` interface of the text document model using the service name "`com.sun.star.style.NumberingStyle`".

The structure of the numbering rules is described in section *7.4.3 Text Documents - Overall Document Features - Line Numbering and Outline Numbering.*

The name of the numbering style is set in the property `NumberingStyleName` of paragraphs (set through the `PropertySet` of a `TextCursor`) or a paragraph style to apply the numbering to the paragraphs.

The following example demonstrates the use of paragraph styles: (Text/TextDocuments.java)

```
/** This method demonstrates how to create, insert and apply styles
 */
protected void StylesExample() {
    try {
        // Go to the end of the document
        mxDocCursor.gotoEnd(false);

        // Insert two paragraph breaks
        mxDocText.insertControlCharacter(
            mxDocCursor, ControlCharacter.PARAGRAPH_BREAK, false);
        mxDocText.insertControlCharacter(
            mxDocCursor, ControlCharacter.PARAGRAPH_BREAK, false);

        // Create a new style from the document's factory
        XStyle xStyle = (XStyle) UnoRuntime.queryInterface(
            XStyle.class, mxDocFactory.createInstance("com.sun.star.style.ParagraphStyle"));

        // Access the XPropertySet interface of the new style
        XPropertySet xStyleProps = (XPropertySet) UnoRuntime.queryInterface(
            XPropertySet.class, xStyle);

        // Give the new style a light blue background
        xStyleProps.setPropertyValue ("ParaBackColor", new Integer(13421823));

        // Get the StyleFamiliesSupplier interface of the document
        XStyleFamiliesSupplier xSupplier = (XStyleFamiliesSupplier)UnoRuntime.queryInterface(
            XStyleFamiliesSupplier.class, mxDoc);

        // Use the StyleFamiliesSupplier interface to get the XNameAccess interface of the
        // actual style families
        XNameAccess xFamilies = (XNameAccess) UnoRuntime.queryInterface (
            XNameAccess.class, xSupplier.getStyleFamilies());

        // Access the 'ParagraphStyles' Family
        XNameContainer xFamily = (XNameContainer) UnoRuntime.queryInterface(
            XNameContainer.class, xFamilies.getByName("ParagraphStyles"));

        // Insert the newly created style into the ParagraphStyles family
        xFamily.insertByName ("All-Singing All-Dancing Style", xStyle);

        // Get the XParagraphCursor interface of the document cursor
        XParagraphCursor xParaCursor = (XParagraphCursor) UnoRuntime.queryInterface(
            XParagraphCursor.class, mxDocCursor);

        // Select the first paragraph inserted
        xParaCursor.gotoPreviousParagraph(false);
        xParaCursor.gotoPreviousParagraph(true);

        // Access the property set of the cursor selection
        XPropertySet xCursorProps = (XPropertySet) UnoRuntime.queryInterface(
            XPropertySet.class, mxDocCursor);

        // Set the style of the cursor selection to our newly created style
        xCursorProps.setPropertyValue("ParaStyleName", "All-Singing All-Dancing Style");

        // Go back to the end
        mxDocCursor.gotoEnd(false);

        // Select the last paragraph in the document
        xParaCursor.gotoNextParagraph(true);

        // And reset it's style to 'Standard' (the programmatic name for the default style)
        xCursorProps.setPropertyValue("ParaStyleName", "Standard");

    } catch (Exception e) {
        e.printStackTrace (System.out);
    }
}
```

7.4.2 Settings

General Document Information

Text documents offer general information about the document through their `com.sun.star.document.XDocumentInfoSupplier` interface. The `DocumentInfo` is a common OpenOffice.org feature and is discussed in *6 Office Development*.

The XDocumentInfoSupplier has one single method:

```
com::sun::star::document::XDocumentInfo getDocumentInfo()
```

which returns a `com.sun.star.document.DocumentInfo` service, offering the statistical information about the document that is available through **File - Properties** in the GUI.

Document Properties

The model implements a `com.sun.star.beans.XPropertySet` that provides properties concerning character formatting and general settings.

The properties for character attributes are `CharFontName`,`CharFontStyleName`, `CharFontFamily`, `CharFontCharSet`, `CharFontPitch` and their Asian counterparts `CharFontStyleNameAsian`, `CharFontFamilyAsian`, `CharFontCharSetAsian`, `CharFontPitchAsian`.

The following properties handle general settings:

Properties of `com.sun.star.text.TextDocument`	
`CharLocale`	`com.sun.star.lang.Locale`. Default locale of the document.
`CharacterCount`	`long` — Number of characters.
`ParagraphCount`	`long` — Number of paragraphs.
`WordCount`	`long` — Number of words.
`WordSeparator`	`string` — Contains all that characters that are treated as separators between words to determine word count.
`RedlineDisplayType`	`short` — Displays redlines as defined in `com.sun.star.document.RedlineDisplayType`.
`RecordChanges`	`boolean` — Determines if redlining is switched on.
`ShowChanges`	`boolean` — Determines if redlines are displayed.
`RedlineProtectionKey`	`sequence < byte >`. Contains the password key.
`ForbiddenCharacters`	`com.sun.star.i18n.ForbiddenCharacters`. Contains characters that are not allowed to be at the first or last character of a text line.
`TwoDigitYear`	`short` — Determines the start of the range, for example, when entering a two-digit year.
`IndexAutoMarkFileURL`	`string` — The URL to the file that contains the search words and settings of the automatic marking of index marks for alphabetical indexes.
`AutomaticControlFocus`	`boolean` — If `true`, the first form object is selected when the document is loaded.
`ApplyFormDesignMode`	`boolean` — Determines if form (database) controls are in the design mode.
`HideFieldTips`	`boolean` — If `true`, the automatic tips displayed for some types of text fields are suppressed.

Creating Default Settings

The com.sun.star.lang.XMultiServiceFactory implemented at the model provides the service com.sun.star.text.Defaults. Use this service to find out default values to set paragraph and character properties of the document to default.

Creating Document Settings

Another set of properties can be created by the service name com.sun.star.document.Settings that contains a number of additional settings.

7.4.3 Line Numbering and Outline Numbering

OpenOffice.org provides automatic numbering for texts. For instance, paragraphs can be numbered or listed with bullets in a hierarchical manner, chapter headings can be numbered and lines can be counted and numbered.

Paragraph and Outline Numbering

com.sun.star.text.NumberingRulesThe key for paragraph numbering is the paragraph property NumberingRules. This property is provided by paragraphs and numbering styles and is a member of com.sun.star.style.ParagraphProperties.

A similar object controls outline numbering and is returned from the method:

```
com::sun::star::container::XIndexReplace getChapterNumberingRules()
```

at the com.sun.star.text.XChapterNumberingSupplier interface that is implemented at the document model.

These objects provide an interface com.sun.star.container.XIndexReplace. Each element of the container represents a numbering level. The writer document provides ten numbering levels. The highest level is zero. Each level of the container consists of a sequence of com.sun.star.beans.PropertyValue.

The two related objects differ in some of properties they provide.

Both of them provide the following properties:

Common Properties for Paragraph and Outline Numbering in com.sun.star.text.NumberingLevel	
Adjust	short — Adjustment of the numbering symbol defined in com.sun.star.text.HoriOrientation.
ParentNumbering	short — Determines if higher numbering levels are included in the numbering, for example, 2.3.1.2.
Prefix Suffix	string — Contains strings that surround the numbering symbol, for example, brackets.
CharStyleName	string — Name of the character style that is applied to the number symbol.
StartWith	short — Determines the value the numbering starts with. The default is one.

Common Properties for Paragraph and Outline Numbering in com.sun.star.text.NumberingLevel	
FirstLineOffset LeftMargin	long — Influences the left indent and left margin of the numbering.
SymbolTextDistance	[optional] long — Distance between the numbering symbol and the text of the paragraph.
NumberingType	short — Determines the type of the numbering defined in com.sun.star.style.NumberingType.

Only paragraphs have the following properties in their NumberingRules property:

Paragraph NumberingRules Properties in com.sun.star.text.NumberingLevel	Description
BulletChar	string — Determines the bullet character if the numbering type is set to NumberingType::CHAR_SPECIAL.
BulletFontName	string — Determines the bullet font if the numbering type is set to NumberingType::CHAR_SPECIAL.
GraphicURL	string — Determines the type, size and orientation of a graphic when the numbering type is set to NumberingType::BITMAP.
GraphicBitmap	Undocumented
GraphicSize	Undocumented
VertOrient	short — Vertical orientation of a graphic according to com.sun.star.text.VertOrientation

Only the chapter numbering rules provide the following property:

Property of com.sun.star.text.ChapterNumberingRule	Description
HeadingStyleName	string — Contains the name of the paragraph style that marks a paragraph as a chapter heading.



Note that the NumberingRules service is returned by value like most properties in the OpenOffice.org API, therefore you must get the rules from the XPropertySet, change them and put the NumberingRules object back into the property.

The following is an example for the NumberingRules service: (Text/TextDocuments.java)

```
/** This method demonstrates how to set numbering types and numbering levels using the
        com.sun.star.text.NumberingRules service
 */
protected void NumberingExample() {
    try {
        // Go to the end of the document
        mxDocCursor.gotoEnd(false);
        // Get the RelativeTextContentInsert interface of the document
        XRelativeTextContentInsert xRelative = (XRelativeTextContentInsert)
            UnoRuntime.queryInterface(XRelativeTextContentInsert.class, mxDocText);

        // Use the document's factory to create the NumberingRules service, and get it's
        // XIndexAccess interface
        XIndexAccess xNum = (XIndexAccess) UnoRuntime.queryInterface(XIndexAccess.class,
            mxDocFactory.createInstance("com.sun.star.text.NumberingRules"));

        // Also get the NumberingRule's XIndexReplace interface
        XIndexReplace xReplace = (XIndexReplace) UnoRuntime.queryInterface(
        XIndexReplace.class, xNum);
```

```
        // Create an array of XPropertySets, one for each of the three paragraphs we're about
        // to create
        XPropertySet xParas[] = new XPropertySet[3];
        for (int i = 0 ; i < 3 ; ++ i) {
            // Create a new paragraph
            XTextContent xNewPara = (XTextContent) UnoRuntime.queryInterface(
                XTextContent.class, mxDocFactory.createInstance(
                    "com.sun.star.text.Paragraph"));

            // Get the XPropertySet interface of the new paragraph and put it in our array
            xParas[i] = (XPropertySet) UnoRuntime.queryInterface(
                XPropertySet.class, xNewPara);

            // Insert the new paragraph into the document after the fish section. As it is
            // an insert
            // relative to the fish section, the first paragraph inserted will be below
            // the next two
            xRelative.insertTextContentAfter (xNewPara, mxFishSection);

            // Separate from the above, but also needs to be done three times

            // Get the PropertyValue sequence for this numbering level
            PropertyValue[] aProps = (PropertyValue []) xNum.getByIndex(i);

            // Iterate over the PropertyValue's for this numbering level, looking for the
            // 'NumberingType' property
            for (int j = 0 ; j < aProps.length ; ++j) {
                if (aProps[j].Name.equals ("NumberingType")) {
                    // Once we find it, set it's value to a new type,
                    // dependent on which
                    // numbering level we're currently on
                    switch ( i ) {
                        case 0 : aProps[j].Value = new Short(NumberingType.ROMAN_UPPER);
                            break;
                        case 1 : aProps[j].Value = new Short(NumberingType.CHARS_UPPER_LETTER);
                            break;
                        case 2 : aProps[j].Value = new Short(NumberingType.ARABIC);
                            break;
                    }
                    // Put the updated PropertyValue sequence back into the
                    // NumberingRules service
                    xReplace.replaceByIndex (i, aProps);
                    break;
                }
            }
        }
        // Get the XParagraphCursor interface of our text cursro
        XParagraphCursor xParaCursor = (XParagraphCursor) UnoRuntime.queryInterface(
            XParagraphCursor.class, mxDocCursor);
        // Go to the end of the document, then select the preceding paragraphs
        mxDocCursor.gotoEnd(false);
        xParaCursor.gotoPreviousParagraph false);
        xParaCursor.gotoPreviousParagraph true);
        xParaCursor.gotoPreviousParagraph true);

        // Get the XPropertySet of the cursor's currently selected text
        XPropertySet xCursorProps = (XPropertySet) UnoRuntime.queryInterface(
            XPropertySet.class, mxDocCursor);

        // Set the updated Numbering rules to the cursor's property set
        xCursorProps.setPropertyValue ("NumberingRules", xNum);
        mxDocCursor.gotoEnd(false);

        // Set the first paragraph that was inserted to a numbering level of 2 (thus it will
        // have Arabic style numbering)
        xParas[0].setPropertyValue ("NumberingLevel", new Short ((short) 2));

        // Set the second paragraph that was inserted to a numbering level of 1 (thus it will
        // have 'Chars Upper Letter' style numbering)
        xParas[1].setPropertyValue ("NumberingLevel", new Short((short) 1));

        // Set the third paragraph that was inserted to a numbering level of 0 (thus it will
        // have 'Chars Upper Letter' style numbering)
        xParas[2].setPropertyValue("NumberingLevel", new Short((short) 0));
    } catch (Exception e) {
        e.printStackTrace (System.out);
    }
}
```

Line Numbering

The text document model supports the interface `com.sun.star.text.XLineNumberingProperties`. The provided object has the properties described in the service `com.sun.star.text.LineNumberingProperties`. It is used in conjunction with the paragraph properties `ParaLineNumberCount` and `ParaLineNumberStartValue`.

Number Formats

The text document model provides access to the number formatter through aggregation, that is, it provides the interface `com.sun.star.util.XNumberFormatsSupplier` seamlessly.

The number formatter is used to format numerical values. For details, refer to *6.2.5 Office Development - Common Application Features - Number Formats*.

In text, text fields with numeric content and table cells provide a property `NumberFormat` that contains a long value that refers to a number format.

7.4.4 Text Sections

A text section is a range of complete paragraphs that can have its own format settings and source location, separate from the surrounding text. Text sections can be nested in a hierarchical structure.

For example, a section is formatted to have text columns that different column settings in a text on a paragraph by paragraph basis. The content of a section can be linked through file links or over a DDE connection.

The text sections support the service `com.sun.star.text.TextSection`. To access the sections, the text document model implements the interface `com.sun.star.text.XTextSectionsSupplier` that provides an interface `com.sun.star.container.XNameAccess` . The returned objects support the interface `com.sun.star.container.XIndexAccess`, as well.

Master documents implement the structure of sub documents using linked text sections.

An example demonstrating the creation, insertion and linking of text sections:
(Text/TextDocuments.java)

```
/** This method demonstrates how to create linked and unlinked sections
 */
protected void TextSectionExample() {
    try {
        // Go to the end of the document
        mxDocCursor.gotoEnd(false);
        // Insert two paragraph breaks
        mxDocText.insertControlCharacter(
            mxDocCursor, ControlCharacter.PARAGRAPH_BREAK, false);
        mxDocText.insertControlCharacter(
        mxDocCursor, ControlCharacter.PARAGRAPH_BREAK, true);

        // Create a new TextSection from the document factory and access it's XNamed interface
        XNamed xChildNamed = (XNamed) UnoRuntime.queryInterface(
            XNamed.class, mxDocFactory.createInstance("com.sun.star.text.TextSection"));
        // Set the new sections name to 'Child_Section'
        xChildNamed.setName("Child_Section");

        // Access the Child_Section's XTextContent interface and insert it into the document
        XTextContent xChildSection = (XTextContent) UnoRuntime.queryInterface(
            XTextContent.class, xChildNamed);
        mxDocText.insertTextContent (mxDocCursor, xChildSection, false);

        // Access the XParagraphCursor interface of our text cursor
        XParagraphCursor xParaCursor = (XParagraphCursor) UnoRuntime.queryInterface(
            XParagraphCursor.class, mxDocCursor);
```

```
        // Go back one paragraph (into Child_Section)
        xParaCursor.gotoPreviousParagraph(false);

        // Insert a string into the Child_Section
        mxDocText.insertString(mxDocCursor, "This is a test", false);

        // Go to the end of the document
        mxDocCursor.gotoEnd(false);

        // Go back two paragraphs
        xParaCursor.gotoPreviousParagraph (false);
        xParaCursor.gotoPreviousParagraph (false);
        // Go to the end of the document, selecting the two paragraphs
        mxDocCursor.gotoEnd(true);

        // Create another text section and access it's XNamed interface
        XNamed xParentNamed = (XNamed) UnoRuntime.queryInterface(XNamed.class,
            mxDocFactory.createInstance("com.sun.star.text.TextSection"));

        // Set this text section's name to Parent_Section
        xParentNamed.setName ("Parent_Section");

        // Access the Parent_Section's XTextContent interface ...
        XTextContent xParentSection = (XTextContent) UnoRuntime.queryInterface(
            XTextContent.class, xParentNamed);
        // ...and insert it into the document
        mxDocText.insertTextContent(mxDocCursor, xParentSection, false);

        // Go to the end of the document
        mxDocCursor.gotoEnd (false);
        // Insert a new paragraph
        mxDocText.insertControlCharacter(
            mxDocCursor, ControlCharacter.PARAGRAPH_BREAK, false);
        // And select the new pargraph
        xParaCursor.gotoPreviousParagraph(true);

        // Create a new Text Section and access it's XNamed interface
        XNamed xLinkNamed = (XNamed) UnoRuntime.queryInterface(
            XNamed.class, mxDocFactory.createInstance("com.sun.star.text.TextSection"));
        // Set the new text section's name to Linked_Section
        xLinkNamed.setName("Linked_Section");

        // Access the Linked_Section's XTextContent interface
        XTextContent xLinkedSection = (XTextContent) UnoRuntime.queryInterface(
            XTextContent.class, xLinkNamed);
        // And insert the Linked_Section into the document
        mxDocText.insertTextContent(mxDocCursor, xLinkedSection, false);

        // Access the Linked_Section's XPropertySet interface
        XPropertySet xLinkProps = (XPropertySet) UnoRuntime.queryInterface(
            XPropertySet.class, xLinkNamed);
        // Set the linked section to be linked to the Child_Section
        xLinkProps.setPropertyValue("LinkRegion", "Child_Section");

        // Access the XPropertySet interface of the Child_Section
        XPropertySet xChildProps = (XPropertySet) UnoRuntime.queryInterface(
            XPropertySet.class, xChildNamed);
        // Set the Child_Section's background colour to blue
        xChildProps.setPropertyValue("BackColor", new Integer(13421823));

        // Refresh the document, so the linked section matches the Child_Section
        XRefreshable xRefresh = (XRefreshable) UnoRuntime.queryInterface(
            XRefreshable.class, mxDoc);
        xRefresh.refresh();
    } catch (Exception e) {
        e.printStackTrace (System.out);
    }
}
```

7.4.5 Page Layout

A page layout in OpenOffice.org is always a page style. A page can not be hard formatted. To change the current page layout, retrieve the current page style from the text cursor property `PageStyleName` and get this page style from the `StyleFamily` `PageStyles`.

Changes of the page layout happen through the properties described in `com.sun.star.style.PageProperties`. Refer to the API reference for details on all the possible properties, including the header and footer texts which are part of these properties.

As headers or footers are connected to a page style, the text objects are provided as properties of the style. Depending on the setting of the page layout, there is one header and footer text object per style available or there are two, a left and right header, and footer text:.

com.sun.star.style.PageProperties containing Headers and Footers	**Description**
HeaderText	com.sun.star.text.Text
HeaderTextLeft	com.sun.star.text.Text
HeaderTextRight	com.sun.star.text.Text
FooterText	com.sun.star.text.Text
FooterTextLeft	com.sun.star.text.Text
FooterTextRight	com.sun.star.text.Text

The page layout of a page style can be equal on left and right pages, mirrored, or separate for right and left pages. This is controlled by the property `PageStyleLayout` that expects values from the enum `com.sun.star.style.PageStyleLayout`. As long as left and right pages are equal, `HeaderText` and `HeaderRightText` are identical. The same applies to the footers.

The text objects in headers and footers are only available if headers or footers are switched on, using the properties `HeaderIsOn` and `FooterIsOn`.

Drawing objects cannot be inserted into headers or footers.

7.4.6 Columns

Text frames, text sections and page styles can be formatted to have columns. The width of columns is relative since the absolute width of the object is unknown in the model. The layout formatting is responsible for calculating the actual widths of the columns.

Columns are applied using the property `TextColumns`. It expects a `com.sun.star.text.TextColumns` service that has to be created by the document factory. The interface `com.sun.star.text.XTextColumns` refines the characteristics of the text columns before applying the created `TextColumns` service to the property `TextColumns`.

Consider the following example to see how to work with text columns:
(Text/TextDocuments.java)

```
/** This method demonstrates the XTextColumns interface and how to insert a blank paragraph
    using the XRelativeTextContentInsert interface
 */
protected void TextColumnsExample() {
    try {
        // Go to the end of the doucment
        mxDocCursor.gotoEnd(false);
        // insert a new paragraph
        mxDocText.insertControlCharacter(mxDocCursor, ControlCharacter.PARAGRAPH_BREAK, false);

        // insert the string 'I am a fish.' 100 times
        for (int i = 0 ; i < 100 ; ++i) {
            mxDocText.insertString(mxDocCursor, "I am a fish.", false);
        }
        // insert a paragraph break after the text
        mxDocText.insertControlCharacter(mxDocCursor, ControlCharacter.PARAGRAPH_BREAK, false);

        // Get the XParagraphCursor interface of our text cursor
        XParagraphCursor xParaCursor = (XParagraphCursor) UnoRuntime.queryInterface(
            XParagraphCursor.class, mxDocCursor);
        // Jump back before all the text we just inserted
        xParaCursor.gotoPreviousParagraph(false);
        xParaCursor.gotoPreviousParagraph(false);

        // Insert a string at the beginning of the block of text
        mxDocText.insertString(mxDocCursor, "Fish section begins:", false);
```

```
        // Then select all of the text
        xParaCursor.gotoNextParagraph(true);
        xParaCursor.gotoNextParagraph(true);

        // Create a new text section and get it's XNamed interface
        XNamed xSectionNamed = (XNamed) UnoRuntime.queryInterface(
            XNamed.class, mxDocFactory.createInstance("com.sun.star.text.TextSection"));

        // Set the name of our new section (appropiately) to 'Fish'
        xSectionNamed.setName("Fish");

        // Create the TextColumns service and get it's XTextColumns interface
        XTextColumns xColumns = (XTextColumns) UnoRuntime.queryInterface(
            XTextColumns.class, mxDocFactory.createInstance("com.sun.star.text.TextColumns"));

        // We want three columns
        xColumns.setColumnCount((short) 3);

        // Get the TextColumns, and make the middle one narrow with a larger margin
        // on the left than the right
        TextColumn[]  aSequence = xColumns.getColumns ();
        aSequence[1].Width /= 2;
        aSequence[1].LeftMargin = 350;
        aSequence[1].RightMargin = 200;
        // Set the updated TextColumns back to the XTextColumns
        xColumns.setColumns(aSequence);

        // Get the property set interface of our 'Fish' section
        XPropertySet xSectionProps = (XPropertySet) UnoRuntime.queryInterface(
            XPropertySet.class, xSectionNamed);

        // Set the columns to the Text Section
        xSectionProps.setPropertyValue("TextColumns", xColumns);

        // Get the XTextContent interface of our 'Fish' section
        mxFishSection = (XTextContent) UnoRuntime.queryInterface(
            XTextContent.class, xSectionNamed);

        // Insert the 'Fish' section over the currently selected text
        mxDocText.insertTextContent(mxDocCursor, mxFishSection, true);

        // Get the wonderful XRelativeTextContentInsert interface
        XRelativeTextContentInsert xRelative = (XRelativeTextContentInsert)
            UnoRuntime.queryInterface(XRelativeTextContentInsert.class, mxDocText);

        // Create a new empty paragraph and get it's XTextContent interface
        XTextContent xNewPara = (XTextContent) UnoRuntime.queryInterface(XTextContent.class,
            mxDocFactory.createInstance("com.sun.star.text.Paragraph"));

        // Insert the empty paragraph after the fish Text Section
        xRelative.insertTextContentAfter(xNewPara, mxFishSection);
    } catch (Exception e) {
        e.printStackTrace(System.out);
    }
}
```

The text columns property consists of `com.sun.star.text.TextColumn` structs. The `Width` elements of all structs in the `TextColumns` sequence make up a sum, that is provided by the method `getReferenceValue()` of the `XTextColumns` interface. To determine the metric width of an actual column, the reference value and the columns width element have to be calculated using the metric width of the object (page, text frame, text section) and a rule of three, for example:

```
nColumn3Width = aColumns[3].Width / xTextColumns.getReferenceValue() * RealObjectWidth
```

The column margins (`LeftMargin`, and `RightMargin` elements of the struct) are inside of the column. Their values do not influence the column width. They just limit the space available for the column content.

The default column setting in OpenOffice.org creates columns with equal margins at inner columns, and no left margin at the leftmost column and no right margin at the rightmost column. Therefore, the relative width of the first and last column is smaller than those of the inner columns. This causes a limitation of this property: Setting the text columns with equal column content widths and equal margins is only possible when the width of the object (text frame, text section) can be determined. Unfortunately this is impossible when the width of the object depends on its environment itself.

7.4.7 Link targets

The interface `com.sun.star.document.XLinkTargetSupplier` of the document model provides all elements of the document that can be used as link targets. These targets can be used for load URLs and sets the selection to a certain position object inside of a document. An example of a URL containing a link target is *"file:///c:/documents/document1|bookmarkname"*.

This interface is used from the hyperlink dialog to detect the links available inside of a document.

The interface `com.sun.star.container.XNameAccess` returned by the method `getLinks()` provides access to an array of target types. These types are:

- Tables
- Text frame
- Graphics
- OLEObjects
- Sections
- Headings
- Bookmarks.

The names of the elements depend on the installed language.

Each returned object supports the interfaces `com.sun.star.beans.XPropertySet` and interface `com.sun.star.container.XNameAccess`. The property set provides the properties `LinkDisplay-Name` (string) and `LinkDisplayBitmap` (`com.sun.star.awt.XBitmap`). Each of these objects provides an array of targets of the relating type. Each target returned supports the interface `com.sun.star.beans.XPropertySet` and the property `LinkDisplayName` (string).

The name of the objects is the bookmark to be added to the document URL, for example, *"Table1|table"*. The `LinkDisplayName` contains the name of the object, e.g. `"Table1"`.

7.5 Text Document Controller

The text document model knows its controller and it can lock the controller to block user interaction. The appropriate methods in the model's `com.sun.star.frame.XModel` interface are:

```
void lockControllers()
void unlockControllers()
boolean hasControllersLocked()
com::sun::star::frame::XController getCurrentController()
void setCurrentController( [in] com::sun::star::frame::XController xController)
```

The controller returned by `getCurrentController()` shares the following interfaces with all other document controllers in OpenOffice.org:

- `com.sun.star.frame.XController`
- `com.sun.star.frame.XDispatchProvider`
- `com.sun.star.ui.XContextMenuInterceptor`

Document controllers are explained in the *6 Office Development*.

7.5.1 TextView

The writer controller implementation supports the interface `com.sun.star.view.XSelectionSupplier` that returns the object that is currently selected in the user interface.

Its method `getSelection()` returns an `any` that may contain the following object depending on the selection:

Selection	**Returned Object**
Text	`com.sun.star.container.XIndexAccess` containing one or more `com.sun.star.uno.XInterface` pointing to a text range.
Selection of table cells	`com.sun.star.uno.XInterface` pointing to a table cursor.
Text frame	`com.sun.star.uno.XInterface` pointing to a text frame.
Graphic object	`com.sun.star.uno.XInterface` pointing to a graphic object.
OLE object	`com.sun.star.uno.XInterface` pointing to an OLE object.
Shape, Form control	`com.sun.star.uno.XInterface` pointing to a `com.sun.star.drawing.ShapeCollection` containing one or more shapes.

- `com.sun.star.view.XControlAccess`
  provides access to the controller of form controls.
- `com.sun.star.text.XTextViewCursorSupplier`
  provides access to the cursor of the view.
- `com.sun.star.text.XRubySelection`
  provides access to rubies contained in the selection. This interface is necessary for Asian language support.
- `com.sun.star.view.XViewSettingsSupplier`
  provides access to the settings of the view as described in the service `com.sun.star.text.ViewSettings`.

Properties of `com.sun.star.text.ViewSettings`	
`ShowAnnotations`	`boolean` — If `true`, annotations (notes) are visible.
`ShowBreaks`	`boolean` — If `true`, paragraph line breaks are visible.
`ShowDrawings`	`boolean` — If `true`, shapes are visible.
`ShowFieldCommands`	`boolean` — If `true`, text fields are shown with their commands, otherwise the content is visible.
`ShowFootnoteBackground`	`boolean` — If `true`, footnotes symbols are displayed with gray background.
`ShowGraphics`	`boolean` — If `true`, graphic objects are visible.
`ShowHiddenParagraphs`	`boolean` — If `true`, hidden paragraphs are displayed.
`ShowHiddenText`	`boolean` — If `true`, hidden text is displayed.
`ShowHoriRuler`	`boolean` — If `true`, the horizontal ruler is displayed.
`ShowHoriScrollBar`	`boolean` — If `true`, the horizontal scroll bar is displayed.
`ShowIndexMarkBackground`	`boolean` — If `true` , index marks are displayed with gray background.
`ShowParaBreaks`	`boolean` — If `true` , paragraph breaks are visible.

ShowProtectedSpaces	boolean — If true, protected spaces (hard spaces) are displayed with gray background.
ShowSoftHyphens	boolean — If true, soft hyphens are displayed with gray background.
ShowSpaces	boolean — If true, spaces are displayed with dots.
ShowTableBoundaries	boolean — If true, table boundaries are displayed.
ShowTables	boolean — If true, tables are visible.
ShowTabstops	boolean — If true, tab stops are visible.
ShowTextBoundaries	boolean — If true, text boundaries are displayed.
ShowTextFieldBackground	boolean — If true, text fields are displayed with gray background.
ShowVertRuler	boolean — If true, the vertical ruler is displayed.
ShowVertScrollBar	boolean — If true, the vertical scroll bar is displayed.
SmoothScrolling	boolean — If true, smooth scrolling is active.
SolidMarkHandles	boolean — If true, handles of drawing objects are visible.
ZoomType	short — defines the zoom type for the document as defined in com.sun.star.view.DocumentZoomType
ZoomValue	short — defines the zoom value to use, the value is given as percentage. Valid only if the property ZoomType is set to com.sun.star.view.DocumentZoomType:BY_VALUE.

In StarOffice 6.0 and OpenOffice.org 1.0 you can only influence the zoom factor by setting the ZoomType to BY_VALUE and adjusting ZoomValue explicitly. The other zoom types have no effect.

7.5.2 TextViewCursor

The text controller has a visible cursor that is used in the GUI. Get the com.sun.star.text.TextViewCursor by calling getTextViewCursor() at the com.sun.star.text.XTextViewCursorSupplier interface of the current text document controller.

It supports the following cursor capabilities that depend on having the necessary information about the current layout state, therefore it is not supported by the model cursor.

com.sun.star.text.XPageCursor

```
boolean jumpToFirstPage()
boolean jumpToLastPage()
boolean jumpToPage( [in] long pageNo)
long getPage()
boolean jumpToNextPage()
boolean jumpToPreviousPage()
boolean jumpToEndOfPage()
boolean jumpToStartOfPage()
```

com.sun.star.view.XScreenCursor

```
boolean screenDown()
boolean screenUp()
```

com.sun.star.view.XLineCursor

```
boolean goDown( [in] long lines,  [in] boolean bExpand)
boolean goUp( [in] long lines,  [in] boolean bExpand)
boolean isAtStartOfLine()
boolean isAtEndOfLine()
void gotoEndOfLine( [in] boolean bExpand)
void gotoStartOfLine( [in] boolean bExpand)
```

`com.sun.star.view.XViewCursor`

```
boolean goLeft( [in] long characters,  [in] boolean bExpand)
boolean goRight( [in] long characters,  [in] boolean bExpand)
boolean goDown( [in] long characters,  [in] boolean bExpand)
boolean goUp( [in] long characters,  [in] boolean bExpand)
```

Additionally the interface `com.sun.star.beans.XPropertySet` is supported.

Currently, the view cursor does not have the capabilities as the document cursor does. Therefore, it is necessary to create a document cursor to have access to the full text cursor functionality. The method `createTextCursorByRange()` is used:

```
XText xCrsrText = xViewCursor.getText();
// Create a TextCursor over the view cursor's contents
XTextCursor xDocumentCursor =     xViewText.createTextCursorByRange(xViewCursor.getStart());
xDocumentCursor.gotoRange(xViewCursor.getEnd(), true);
```

8 Spreadsheet Documents

8.1 Overview

OpenOffice.org API knows three variants of tables: *text tables* (see *[Chapter:TextTables]*), *database tables* (see *[Chapter:DatabaseTables]*) and *spreadsheets*. Each of the table concepts have their own purpose. Text tables handle text contents, database tables offer database functionality and spreadsheets operate on data cells that can be evaluated. Being specialized in such a way means that each concept has its strength. Text tables offer full functionality for text formatting, where spreadsheets support complex calculations. Alternately, spreadsheets support only basic formatting capabilities and text tables perform elementary calculations.

The implementation of the various tables differ due to each of their specializations. Basic table features are defined in the module `com.sun.star.table`. Regarding the compatibility of text and spreadsheet tables, the corresponding features are also located in the module `com.sun.star.table`. In addition, spreadsheet tables are fully based on the specifications given and are extended by additional specifications from the module `com.sun.star.sheet`. Several services of the spreadsheet application representing cells and cell ranges extend the common services from the module com::sun::star::table. The following table shows the services for cells and cell ranges.

Spreadsheet service	**Included com::sun::star::table service**
`com.sun.star.sheet.SheetCell`	`com.sun.star.table.Cell`
`com.sun.star.sheet.Cells`	-
`com.sun.star.sheet.SheetCellRange`	`com.sun.star.table.CellRange`
`com.sun.star.sheet.SheetCellRanges`	-
`com.sun.star.sheet.SheetCellCursor`	`com.sun.star.table.CellCursor`

The spreadsheet document model in the OpenOffice.org API has five major architectural areas (see *Illustration 79*) The five areas are:

- Spreadsheets Container
- Service Manager (document internal)
- DrawPages
- Content Properties
- Objects for Styling

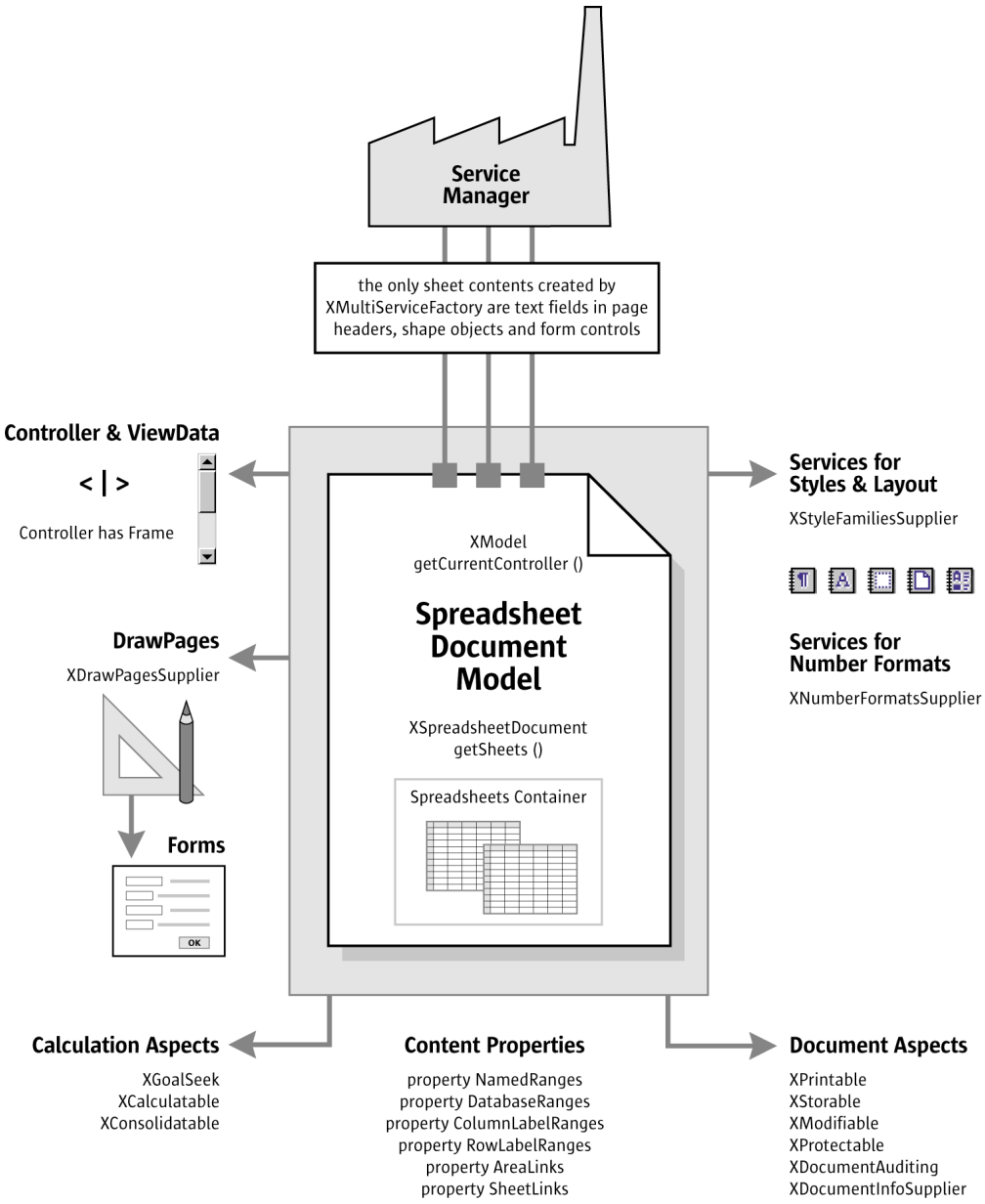


Illustration 78: Spreadsheet Document Component

The core of the spreadsheet document model are the spreadsheets contained in the spreadsheets container. When working with document data, almost everything happens in the spreadsheet objects extracted from the spreadsheets container.

The service manager of the spreadsheet document model creates shape objects, text fields for page headers and form controls that can be added to spreadsheets. Note, that the document service manager is different from the main service manager used when connecting to the office. Each document model has its own service manager, so that the services can be adapted to the document they are required for. For instance, a text field is ordered and inserted into the page header text of a sheet using `com.sun.star.text.XText:insertTextContent()` or the service manager is asked for a shape object and inserts it into a sheet using `add()` at the drawpage.

Each sheet in a spreadsheet document can have a drawpage for drawing contents. A drawpage can be visualized as a transparent layer above a sheet. The spreadsheet model is able to provide all drawpages in a spreadsheet document at once.

Linked and named contents from all sheets are accessed through content properties at the document model. There are no content suppliers as in text documents, because the actual content of a spreadsheet document lies in its sheet objects. Rather, there are only certain properties for named and linked contents in all sheets.

Finally, there are services that allow for document wide styling and structuring of the spreadsheet document. Among them are style family suppliers for cells and pages, and a number formats supplier.

Besides these five architectural areas, there are document and calculation aspects shown at the bottom of the illustration. The document aspects of our model are: it is printable, storable, and modifiable, it can be protected and audited, and it supplies general information about itself. On the lower left of the illustration, the calculation aspects are listed. Although almost all spreadsheet functionality can be found at the spreadsheet objects, a few common functions are bound to the spreadsheet document model: goal seeking, consolidation and recalculation of all cells.

Finally, the document model has a controller that provides access to the graphical user interface of the model and has knowledge about the current view status in the user interface (see the upper left of the illustration).

The usage of the spreadsheet objects in the spreadsheets container is discussed in detail in the section *8.3 Spreadsheet Documents - Working with Spreadsheets*. Before discussing spreadsheet objects, consider two examples and how they handle a spreadsheet document, that is, how to create, open, save and print.

8.1.1 Example: Adding a New Spreadsheet

The following helper method opens a new spreadsheet document component. The method `getRemoteServiceManager()` retrieves a connection. Refer to chapter *2 First Steps* for additional information.

```
import com.sun.star.lang.XComponent;
import com.sun.star.frame.XComponentLoader;
import com.sun.star.beans.PropertyValue;

...

protected XComponent newSpreadsheetComponent() throws java.lang.Exception {
    String loadUrl = "private:factory/scalc";
    xRemoteServiceManager = this.getRemoteServiceManager(unoUrl);
    Object desktop = xRemoteServiceManager.createInstanceWithContext(
        "com.sun.star.frame.Desktop", xRemoteContext);
    XComponentLoader xComponentLoader = (XComponentLoader)UnoRuntime.queryInterface(
        XComponentLoader.class, desktop);
    PropertyValue[] loadProps = new PropertyValue[0];
    return xComponentLoader.loadComponentFromURL(loadUrl, "_blank", 0, loadProps);
}
```

Our helper returns a `com.sun.star.lang.XComponent` interface for the recently loaded document. Now the `XComponent` is passed to the following method `insertSpreadsheet()` to add a new spreadsheet to our document. (Spreadsheet/SpreadsheetDocHelper.java)

```
import com.sun.star.sheet.XSpreadsheetDcoument;
import com.sun.star.sheet.XSpreadsheet;

...

/** Inserts a new empty spreadsheet with the specified name.
    @param xSheetComponent  The XComponent interface of a loaded document object
    @param aName  The name of the new sheet.
    @param nIndex  The insertion index.
    @return  The XSpreadsheet interface of the new sheet.
```

```
 */
public XSpreadsheet insertSpreadsheet(
        XComponent xSheetComponent, String aName, short nIndex) {
    XSpreadsheetDocument xDocument = (XSpreadsheetDocument)UnoRuntime.queryInterface(
        XSpreadsheetDocument.class, xSheetComponent);

    // Collection of sheets
    com.sun.star.sheet.XSpreadsheets xSheets = xDocument.getSheets();
    com.sun.star.sheet.XSpreadsheet xSheet = null;

    try {
        xSheets.insertNewByName(aName, nIndex);
        xSheet = xSheets.getByName(aName);
    } catch (Exception ex) {
    }

    return xSheet;
}
```

8.1.2 Example: Editing Spreadsheet Cells

The method `insertSpreadsheet()` returns a `com.sun.star.sheet.XSpreadsheet` interface. This interface is passed to the method below, which shows how to access and modify the content and formatting of single cells. The interface `com.sun.star.sheet.XSpreadsheet` returned by `insertSpreadsheet()` is derived from `com.sun.star.table.XCellRange`. By working with it, cells can be accessed immediately using `getCellByPosition()`: (Spreadsheet/GeneralTableSample.java)

```
void cellWork(XSpreadsheet xRange) {

    com.sun.star.beans.XPropertySet xPropSet = null;
    com.sun.star.table.XCell xCell = null;

    // Access and modify a VALUE CELL
    xCell = xRange.getCellByPosition(0, 0);
    // Set cell value.
    xCell.setValue(1234);

    // Get cell value.
    double nDblValue = xCell.getValue() * 2;
    xRange.getCellByPosition(0, 1).setValue(nDblValue);

    // Create a FORMULA CELL
    xCell = xRange.getCellByPosition(0, 2);
    // Set formula string.
    xCell.setFormula("=1/0");

    // Get error type.
    boolean bValid = (xCell.getError() == 0);
    // Get formula string.
    String aText = "The formula " + xCell.getFormula() + " is ";
    aText += bValid ? "valid." : "erroneous.";

    // Insert a TEXT CELL using the XText interface
    xCell = xRange.getCellByPosition(0, 3);
    com.sun.star.text.XText xCellText = (com.sun.star.text.XText)
        UnoRuntime.queryInterface(com.sun.star.text.XText.class, xCell);
    com.sun.star.text.XTextCursor xTextCursor = xCellText.createTextCursor();
    xCellText.insertString(xTextCursor, aText, false);
}
```

8.2 Handling Spreadsheet Document Files

8.2.1 Creating and Loading Spreadsheet Documents

If a document in OpenOffice.org API is required, begin by getting a `com.sun.star.frame.Desktop` service from the service manager. The desktop handles all document components in OpenOffice.org API. It is discussed thoroughly in the chapter *6 Office Development*. Office documents are often called components, because they support the

com.sun.star.lang.XComponent interface. An XComponent is a UNO object that can be disposed of directly and broadcast an event to other UNO objects when the object is disposed.

The Desktop can load new and existing components from a URL. For this purpose it has a com.sun.star.frame.XComponentLoader interface that has one single method to load and instantiate components from a URL into a frame:

```
com::sun::star::lang::XComponent loadComponentFromURL( [IN] string aURL,
          [IN] string aTargetFrameName,
          [IN] long nSearchFlags,
          [IN] sequence <com::sun::star::beans::PropertyValue[] aArgs > )
```

The interesting parameters in our context is the URL that describes the resource that is loaded and the load arguments. For the target frame, pass a "_blank" and set the search flags to 0. In most cases, existing frames are not reused.

The URL can be a file: URL, an http: URL, an ftp: URL or a private: URL. Locate the correct URL format in the **Load URL** box in the function bar of OpenOffice.org API. For new spreadsheet documents, a special URL scheme is used. The scheme is "private:", followed by "factory". The resource is "scalc" for OpenOffice.org API spreadsheet documents. For a new spreadsheet document, use "private:factory/scalc".

The load arguments are described in com.sun.star.document.MediaDescriptor. The properties AsTemplate and Hidden are boolean values and used for programming. If AsTemplate is true, the loader creates a new untitled document from the given URL. If it is false, template files are loaded for editing. If Hidden is true, the document is loaded in the background. This is useful to generate a document in the background without letting the user observe what is happening. For instance, use it to generate a document and print it out without previewing. Refer to *6 Office Development* for other available options. This snippet loads a document in hidden mode:

```
// the method getRemoteServiceManager is described in the chapter First Steps
mxRemoteServiceManager = this.getRemoteServiceManager(unoUrl);

// retrieve the Desktop object, we need its XComponentLoader
Object desktop = mxRemoteServiceManager.createInstanceWithContext(
    "com.sun.star.frame.Desktop", mxRemoteContext);

// query the XComponentLoader interface from the Desktop service
XComponentLoader xComponentLoader = (XComponentLoader)UnoRuntime.queryInterface(
    XComponentLoader.class, desktop);

// define load properties according to com.sun.star.document.MediaDescriptor

/* or simply create an empty array of com.sun.star.beans.PropertyValue structs:
   PropertyValue[] loadProps = new PropertyValue[0]
 */

// the boolean property Hidden tells the office to open a file in hidden mode
PropertyValue[] loadProps = new PropertyValue[1];
loadProps[0] = new PropertyValue();
loadProps[0].Name = "Hidden";
loadProps[0].Value = new Boolean(true);
loadUrl = "file:///c:/MyCalcDocument.sxc"

// load
return xComponentLoader.loadComponentFromURL(loadUrl, "_blank", 0, loadProps);
```

8.2.2 Saving Spreadsheet Documents

Storing

Documents are storable through their interface com.sun.star.frame.XStorable. This interface is discussed in detail in *6 Office Development*. An XStorable implements these operations:

```
boolean hasLocation()
string getLocation()
```

```
boolean isReadonly()
void store()
void storeAsURL([in] string aURL, [in] sequence< com::sun::star::beans::PropertyValue > aArgs)
void storeToURL([in] string aURL, [in] sequence< com::sun::star::beans::PropertyValue > aArgs)
```

The method names are evident. The method `storeAsUrl()` is the exact representation of **File – Save As** from the **File** menu, that is, it changes the current document location. In contrast, `storeToUrl()` stores a copy to a new location, but leaves the current document URL untouched.

Exporting

For exporting purposes, a filter name can be passed that triggers an export to other file formats. The property needed for this purpose is the string argument `FilterName` that takes filter names defined in the configuration file:

<OfficePath>\share\config\registry\instance\org\openoffice\Office\TypeDetection.xml

In *TypeDetection.xml* look for `<Filter/>` elements, their `cfg:name` attribute contains the needed strings for `FilterName`. The proper filter name for StarWriter 5.x is "StarWriter 5.0", and the export format "MS Word 97" is also popular. This is the element in *TypeDetection.xml* that describes the MS Excel 97 filter:

```
<Filter cfg:name="MS Excel 97">
   <Installed cfg:type="boolean">true</Installed>
   <UIName cfg:type="string" cfg:localized="true">
     <cfg:value xml:lang="en-US">Microsoft Excel 97/2000/XP</cfg:value>
   </UIName>
   <Data cfg:type="string">5,calc_MS_Excel_97,com.sun.star.sheet.SpreadsheetDocument,,3,,0,,</Data>
 </Filter>
```

The following method stores a document using this filter:

```
/** Store a document, using the MS Excel 97/2000/XP Filter
 */
protected void storeDocComponent(XComponent xDoc, String storeUrl) throws java.lang.Exception {

    XStorable xStorable = (XStorable)UnoRuntime.queryInterface(XStorable.class, xDoc);
    PropertyValue[] storeProps = new PropertyValue[1];
    storeProps[0] = new PropertyValue();
    storeProps[0].Name = "FilterName";
    storeProps[0].Value = "MS Excel 97";
    xStorable.storeAsURL(storeUrl, storeProps);
}
```

If an empty array of `PropertyValue` structs is passed, the native .sxc format of OpenOffice.org API is used.

Filter Options

Loading and saving OpenOffice.org API documents is described in *6.1.5 Office Development - OpenOffice.org Application Environment - Handling Documents*. This section lists all the filter names for spreadsheet documents and describes the filter options for text file import.

The filter name and options are passed on loading or saving a document in a sequence of `com.sun.star.beans.PropertyValues`. The property `FilterName` contains the name and the property `FilterOptions` contains the filter options.



All filter names are case-sensitive. For compatibility reasons the filter names will not be changed. Therefore, some of the filters seem to have “curious” names.

The list of filter names (the last two columns show the possible directions of the filters):

Filter name	Description	Import	Export
StarOffice XML (Calc)	Standard XML filter	•	•
calc_StarOffice_XML_Calc_Template	XML filter for templates	•	•
StarCalc 5.0	The binary format of StarOffice Calc 5.x	•	•
StarCalc 5.0 Vorlage/Template	StarOffice Calc 5.x templates	•	•
StarCalc 4.0	The binary format of StarCalc 4.x	•	•
StarCalc 4.0 Vorlage/Template	StarCalc 4.x templates	•	•
StarCalc 3.0	The binary format of StarCalc 3.x	•	•
StarCalc 3.0 Vorlage/Template	StarCalc 3.x templates	•	•
HTML (StarCalc)	HTML filter	•	•
calc_HTML_WebQuery	HTML filter for external data queries	•	
MS Excel 97	Microsoft Excel 97/2000/XP	•	•
MS Excel 97 Vorlage/Template	Microsoft Excel 97/2000/XP templates	•	•
MS Excel 95	Microsoft Excel 5.0/95	•	•
MS Excel 5.0/95	Different name for the same filter	•	•
MS Excel 95 Vorlage/Template	Microsoft Excel 5.0/95 templates	•	•
MS Excel 5.0/95 Vorlage/Template	Different name for the same filter	•	•
MS Excel 4.0	Microsoft Excel 2.1/3.0/4.0	•	
MS Excel 4.0 Vorlage/Template	Microsoft Excel 2.1/3.0/4.0 templates	•	
Lotus	Lotus 1-2-3	•	
Text - txt - csv (StarCalc)	Comma separated values	•	•
Rich Text Format (StarCalc)		•	•
dBase		•	•
SYLK	Symbolic Link	•	•
DIF	Data Interchange Format	•	•

Filter Options for Lotus, dBase and DIF Filters

These filters accept a string containing the numerical index of the used character set for single-byte characters, that is, 0 for the system character set.

Filter Options for the CSV Filter

This filter accepts an option string containing five tokens, separated by commas. The following table shows an example string for a file with four columns of type date – number – number - number. In the table the tokens are numbered from (1) to (5). Each token is explained below.

Example Filter Options String	Field Separator (1)	Text Delimiter (2)	Character Set (3)	Number of First Line (4)	Cell Format Codes for the four Columns (5)	
					Column	Code
File Format: Four columns date – num – num – num	,	"	System	line no. 1	1 2 3 4	YY/MM/DD = 5 Standard = 1 Standard = 1 Standard = 1
Token	44	34	0	1	1/5/2/1/3/1/4/1	

For the filter options above, set the PropertyValue `FilterOptions` in the load arguments to "44,34,0,1,1/5/2/1/3/1/4/1". There are a number of possible settings for the five tokens.

1. Field separator(s) as ASCII values. Multiple values are separated by the slash sign ("/"), that is, if the values are separated by semicolons and horizontal tabulators, the token would be 59/9. To treat several consecutive separators as one, the four letters `/MRG` have to be appended to the token. If the file contains fixed width fields, the three letters `FIX` are used.
2. The text delimiter as ASCII value, that is, 34 for double quotes and 39 for single quotes.
3. The character set used in the file as described above.
4. Number of the first line to convert. The first line in the file has the number 1.
5. Cell format of the columns. The content of this token depends on the value of the first token.
   - If value separators are used, the form of this token is *column/format*[*/column/format/*...] where *column* is the number of the column, with 1 being the leftmost column. The *format* is explained below.
   - If the first token is `FIX` it has the form *start/format*[*/start/format/*...], where *start* is the number of the first character for this field, with 0 being the leftmost character in a line. The *format* is explained below.

   Format specifies which cell format should be used for a field during import:

Format Code	Meaning
1	Standard
2	Text
3	MM/DD/YY
4	DD/MM/YY
5	YY/MM/DD
6	-
7	-
8	-
9	ignore field (do not import)
10	US-English

The type code 10 indicates that the content of a field is US-English. This is useful if a field contains decimal numbers that are formatted according to the US system (using "." as decimal separator and "," as thousands separator). Using 10 as a format specifier for this

field tells OpenOffice.org API to correctly interpret its numerical content, even if the decimal and thousands separator in the current language are different.

8.2.3 Printing Spreadsheet Documents

Printer and Print Job Settings

Printing is a common office functionality. The chapter *6 Office Development* provides in-depth information about it. The spreadsheet document implements the `com.sun.star.view.XPrintable` interface for printing. It consists of three methods:

```
sequence< com::sun::star::beans::PropertyValue > getPrinter()
void setPrinter([in] sequence< com::sun::star::beans::PropertyValue > aPrinter)
void print([in] sequence< com::sun::star::beans::PropertyValue > xOptions)
```

The following code is used with a given document xDoc to print to the standard printer without any settings:

```
// query the XPrintable interface from your document
XPrintable xPrintable = (XPrintable)UnoRuntime.queryInterface(XPrintable.class, xDoc);

// create an empty printOptions array
PropertyValue[] printOpts = new PropertyValue[0];

// kick off printing
xPrintable.print(printOpts);
```

There are two groups of properties involved in general printing. The first one is used with `setPrinter()` and `getPrinter()`, and controls the printer, and the second is passed to `print()` and controls the print job.

`com.sun.star.view.PrinterDescriptor` comprises the properties for the printer:

Properties of `com.sun.star.view.PrinterDescriptor`	
`Name`	string — Specifies the name of the printer queue to be used.
`PaperOrientation`	`com.sun.star.view.PaperOrientation` Specifies the orientation of the paper.
`PaperFormat`	`com.sun.star.view.PaperFormat` Specifies a predefined paper size or if the paper size is a user-defined size.
`PaperSize`	`com.sun.star.awt.Size` Specifies the size of the paper in 100th mm.
`IsBusy`	boolean — Indicates if the printer is busy.
`CanSetPaperOrientation`	boolean — Indicates if the printer allows changes to PaperOrientation.
`CanSetPaperFormat`	boolean — Indicates if the printer allows changes to PaperFormat.
`CanSetPaperSize`	boolean — Indicates if the printer allows changes to PaperSize.

`com.sun.star.view.PrintOptions` contains the following possibilities for a print job:

Properties of `com.sun.star.view.PrintOptions`	
`CopyCount`	short — Specifies the number of copies to print.
`FileName`	string — If set, specifies the name of the file to print to.
`Collate`	boolean — Advises the printer to collate the pages of the copies. If true, a whole document is printed prior to the next copy, otherwise the page copies are completed together.

`Sort`	boolean — Advises the printer to sort the pages of the copies.
`Pages`	string — Specifies the pages to print with the same format as in the print dialog of the GUI, for example, "1, 3, 4-7, 9-".

The following method uses `PrinterDescriptor` and `PrintOptions` to print to a special printer, and preselect the pages to print.

```
protected void printDocComponent(XComponent xDoc) throws java.lang.Exception {

    XPrintable xPrintable = (XPrintable)UnoRuntime.queryInterface(XPrintable.class, xDoc);
    PropertyValue[] printerDesc = new PropertyValue[1];
    printerDesc[0] = new PropertyValue();
    printerDesc[0].Name = "Name";
    printerDesc[0].Value = "5D PDF Creator";

    xPrintable.setPrinter(printerDesc);

    PropertyValue[] printOpts = new PropertyValue[1];
    printOpts[0] = new PropertyValue();
    printOpts[0].Name = "Pages";
    printOpts[0].Value = "3-5,7";

    xPrintable.print(printOpts);
}
```

Page Breaks and Scaling for Printout

Manual page breaks can be inserted and removed using the property `IsStartOfNewPage` of the services `com.sun.star.table.TableColumn` and `com.sun.star.table.TableRow`. For details, refer to the section about page breaks in the chapter *8 Spreadsheet Documents*.

To reduce the page size of a sheet so that the sheet fits on a fixed number of printout pages, use the properties `PageScale` and `ScaleToPages` of the current page style. Both of the properties are `short` numbers. The `PageScale` property expects a percentage and `ScaleToPages` is the number of pages the printout is to fit. The page style is available through the interface `com.sun.star.style.XStyleFamiliesSupplier` of the document component, and is described in the chapter *8.4.1 Spreadsheet Documents - Overall Document Features - Styles.*

Print Areas

The Interface `com.sun.star.sheet.XPrintAreas` is available at spreadsheets. It provides access to the addresses of all printable cell ranges, represented by a sequence of `com.sun.star.table.CellRangeAddress` structs.

Methods of `com.sun.star.sheet.XPrintAreas`	
`getPrintAreas()`	Returns the print areas of the sheet.
`setPrintAreas()`	Sets the print areas of the sheet.
`getPrintTitleColumns()`	Returns true if the title columns are repeated on all subsequent print pages to the right.
`setPrintTitleColumns()`	Specifies if the title columns are repeated on all subsequent print pages to the right.
`getTitleColumns()`	Returns the range of columns that are marked as title columns.
`setTitleColumns()`	Sets the range of columns marked as title columns.
`getPrintTitleRows()`	Returns true if the title rows are repeated on all subsequent print pages to the bottom.

Methods of `com.sun.star.sheet.XPrintAreas`	
`setPrintTitleRows()`	Specifies if the title rows are repeated on all subsequent print pages to the bottom.
`getTitleRows()`	Returns the range of rows that are marked as title rows.
`setTitleRows()`	Sets the range of rows marked as title rows.

8.3 Working with Spreadsheet Documents

8.3.1 Document Structure

Spreadsheet Document

The whole spreadsheet document is represented by the service `com.sun.star.sheet.SpreadsheetDocument`. It implements interfaces that provide access to the container of spreadsheets and methods to modify the document wide contents, for instance, data consolidation.

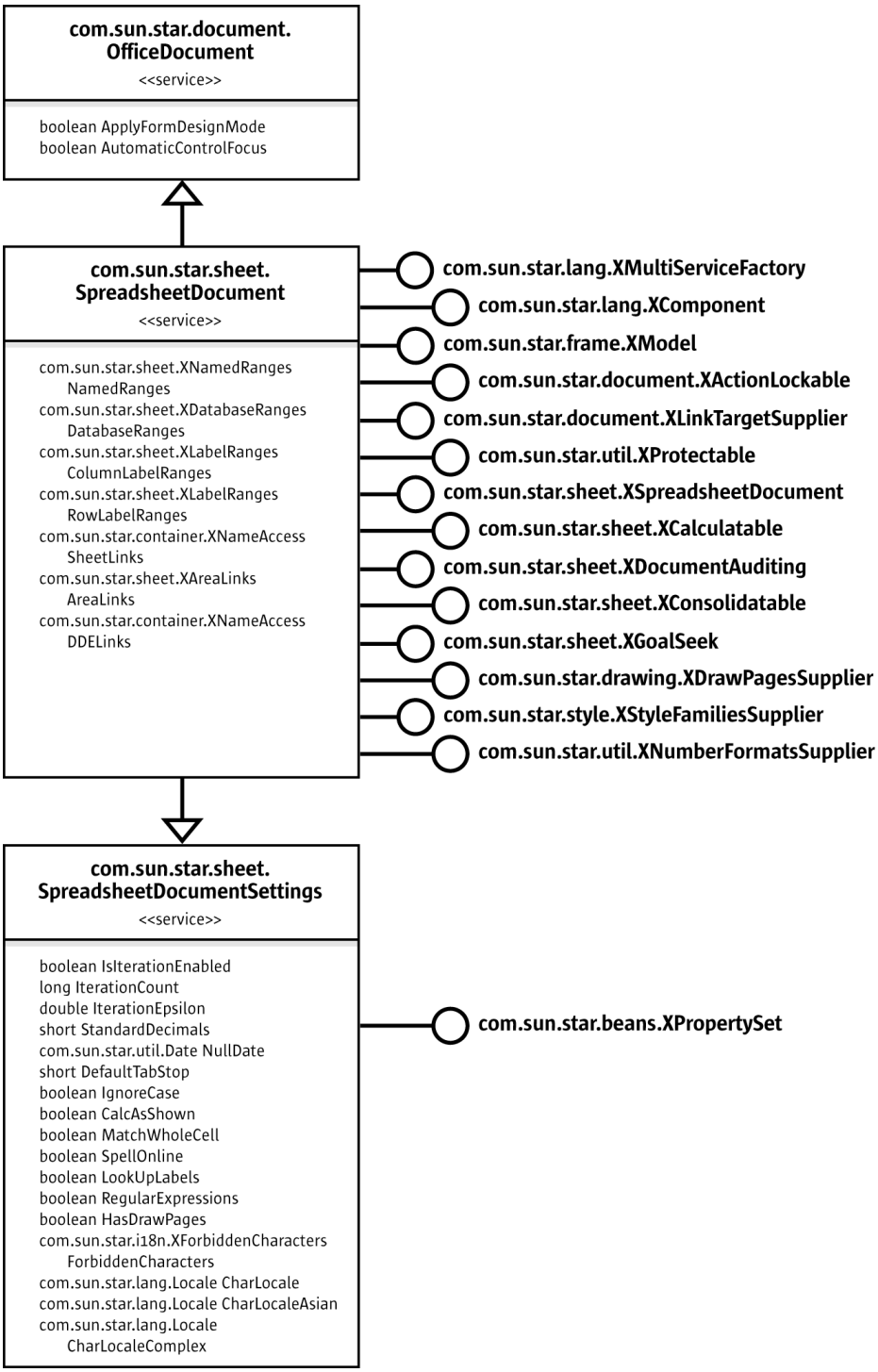


Illustration 79: Spreadsheet Document

A spreadsheet document contains a collection of spreadsheets with at least one spreadsheet, represented by the service `com.sun.star.sheet.Spreadsheets`. The method `getSheets()` of the Interface `com.sun.star.sheet.XSpreadsheetDocument` returns the interface `com.sun.star.sheet.XSpreadsheets` for accessing the container of sheets.

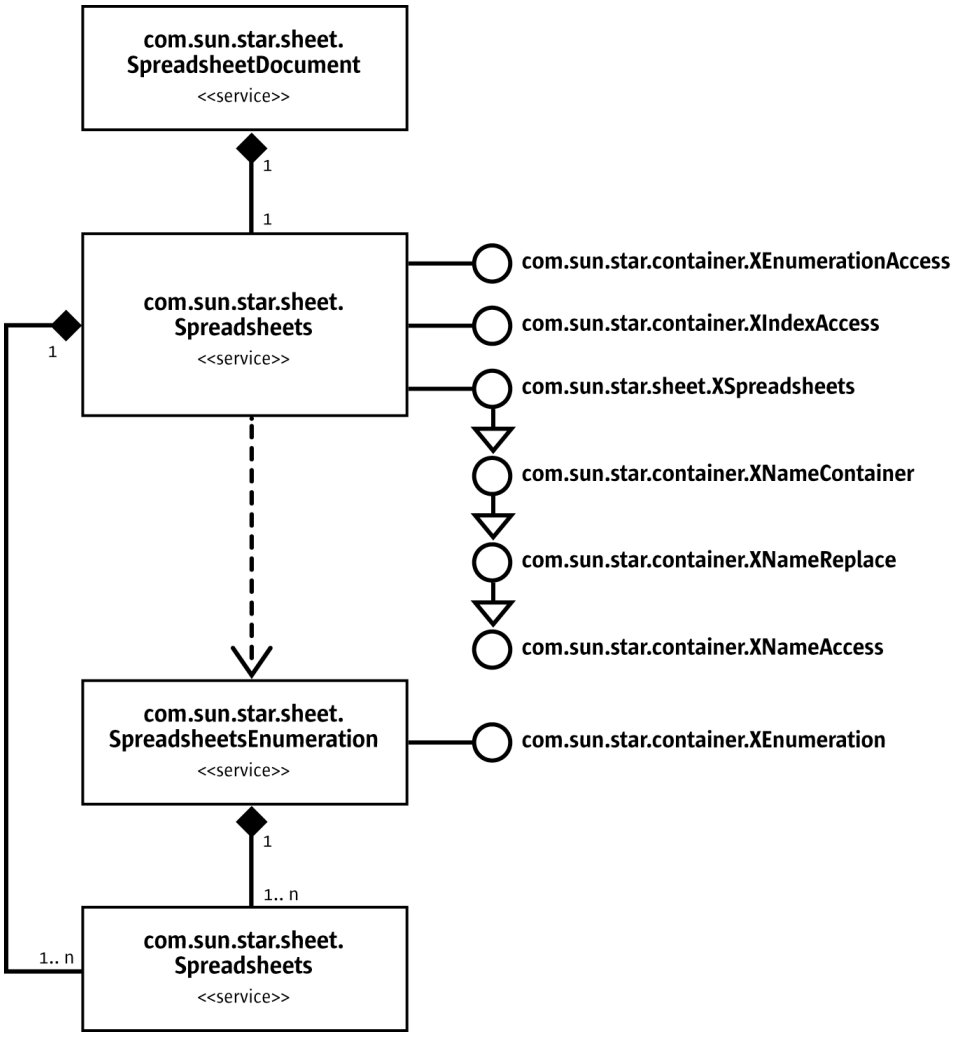


Illustration 80: Spreadsheets Container

When the spreadsheet container is retrieved from a document using its `getSheets()` method, it is possible to access the sheets in three different ways:

by index
: Using the interface `com.sun.star.container.XIndexAccess` allows access to spreadsheets by their index.

with an enumeration
: Using the service `com.sun.star.sheet.SpreadsheetsEnumeration` spreadsheets can be accessed as an enumeration.

by name
: The interface `com.sun.star.sheet.XSpreadsheets` is derived from `com.sun.star.container.XNameContainer` and therefore contains all methods for accessing the sheets with a name. It is possible to *get* a spreadsheet using `com.sun.star.container.XNameAccess`) to *replace* it with another sheet (interface `com.sun.star.container.XNameReplace`), and to *insert* and *remove* a spreadsheet (interface `com.sun.star.container.XNameContainer`).

The following two helper methods demonstrate how spreadsheets are accessed by their indexes and their names: (Spreadsheet/SpreadsheetDocHelper.java)

```
/** Returns the spreadsheet with the specified index (0-based).
    @param xDocument  The XSpreadsheetDocument interface of the document.
    @param nIndex  The index of the sheet.
    @return  The XSpreadsheet interface of the sheet. */
public com.sun.star.sheet.XSpreadsheet getSpreadsheet(
        com.sun.star.sheet.XSpreadsheetDocument xDocument, int nIndex) {

    // Collection of sheets
    com.sun.star.sheet.XSpreadsheets xSheets = xDocument.getSheets();
    com.sun.star.sheet.XSpreadsheet xSheet = null;

    try {
        com.sun.star.container.XIndexAccess xSheetsIA = (com.sun.star.container.XIndexAccess)
            UnoRuntime.queryInterface(com.sun.star.container.XIndexAccess.class, xSheets);
        xSheet = (com.sun.star.sheet.XSpreadsheet) xSheetsIA.getByIndex(nIndex);
    } catch (Exception ex) {
    }

    return xSheet;
}
```

```
/** Returns the spreadsheet with the specified name.
    @param xDocument  The XSpreadsheetDocument interface of the document.
    @param aName  The name of the sheet.
    @return  The XSpreadsheet interface of the sheet. */
public com.sun.star.sheet.XSpreadsheet getSpreadsheet(
        com.sun.star.sheet.XSpreadsheetDocument xDocument,
        String aName) {

    // Collection of sheets
    com.sun.star.sheet.XSpreadsheets xSheets = xDocument.getSheets();
    com.sun.star.sheet.XSpreadsheet xSheet = null;

    try {
        com.sun.star.container.XNameAccess xSheetsNA = (com.sun.star.container.XNameAccess)
            UnoRuntime.queryInterface(com.sun.star.container.XNameAccess.class, xSheets);
        xSheet = (com.sun.star.sheet.XSpreadsheet) xSheetsNA.getByName(aName);
    } catch (Exception ex) {
    }

    return xSheet;
}
```

The interface `com.sun.star.sheet.XSpreadsheets` contains additional methods that use the name of spreadsheets to add new sheets, and to move and copy them:

Methods of `com.sun.star.sheet.XSpreadsheets`	
`insertNewByName()`	Creates a new empty spreadsheet with the specified name and inserts it at the specified position.
`moveByName()`	Moves the spreadsheet with the specified name to a new position.
`copyByName()`	Creates a copy of a spreadsheet, renames it and inserts it at a new position.

The method below shows how a new spreadsheet is inserted into the spreadsheet collection of a document with the specified name. (Spreadsheet/SpreadsheetDocHelper.java)

```
/** Inserts a new empty spreadsheet with the specified name.
    @param xDocument  The XSpreadsheetDocument interface of the document.
    @param aName  The name of the new sheet.
    @param nIndex  The insertion index.
    @return  The XSpreadsheet interface of the new sheet.
 */
public com.sun.star.sheet.XSpreadsheet insertSpreadsheet(
        com.sun.star.sheet.XSpreadsheetDocument xDocument,
        String aName, short nIndex ) {
    // Collection of sheets
    com.sun.star.sheet.XSpreadsheets xSheets = xDocument.getSheets();
    com.sun.star.sheet.XSpreadsheet xSheet = null;

    try {
        xSheets.insertNewByName(aName, nIndex);
        xSheet = xSheets.getByName( aName );
    } catch (Exception ex) {
    }

    return xSheet;
}
```

Spreadsheet Services - Overview

The previous section introduced the organization of the spreadsheets in a document and how they can be handled. This section discusses the spreadsheets themselves. The following illustration provides an overview about the main API objects that can be used in a spreadsheet.

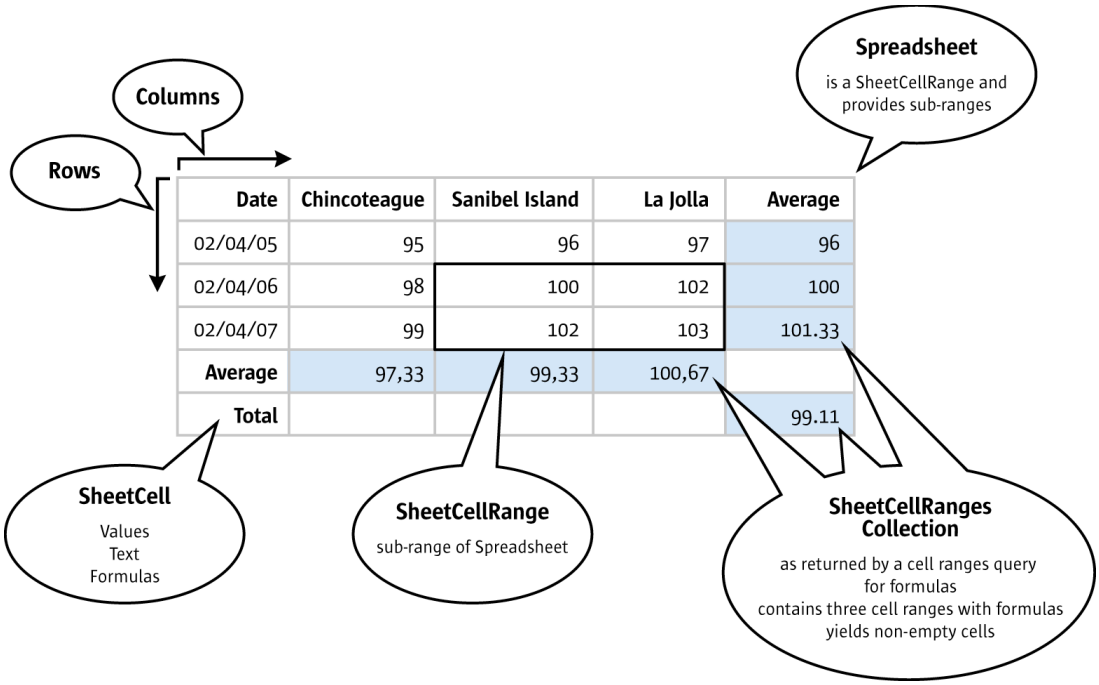


Illustration 81: Main Spreadsheet Services

The main services in a spreadsheet are `com.sun.star.sheet.Spreadsheet`, `com.sun.star.sheet.SheetCellRange`, the cell service `com.sun.star.sheet.SheetCell`, the collection of cell ranges `com.sun.star.sheet.SheetCellRanges` and the services `com.sun.star.table.TableColumn` and `com.sun.star.table.TableRow`. An overview of the capabilities of these services is provided below.

Capabilities of Spreadsheet

The spreadsheet is a `com.sun.star.sheet.Spreadsheet` service that includes the service `com.sun.star.sheet.SheetCellRange`, that is, a spreadsheet is a cell range with additional capabilities concerning the entire sheet:

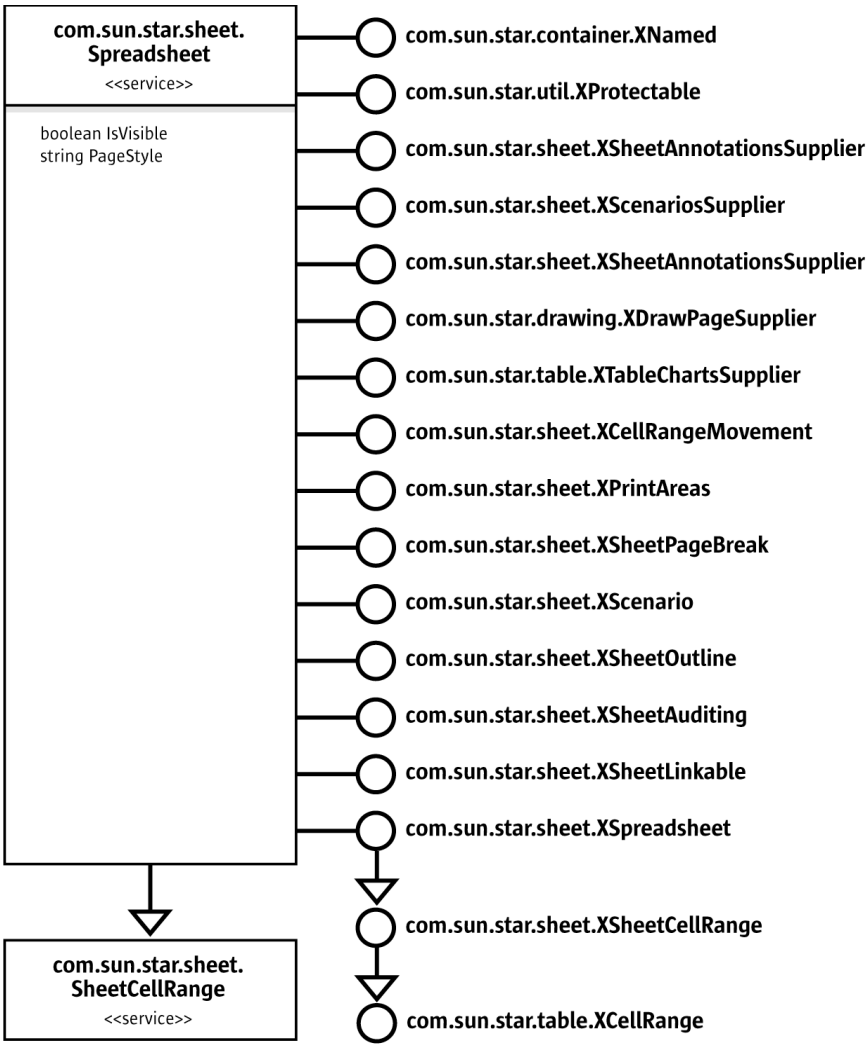


Illustration 82: Spreadsheet

- It can be named using `com.sun.star.container.XNamed`.
- It has interfaces for sheet analysis. Data pilot tables, sheet outlining, sheet auditing (detective) and scenarios all are controlled from the spreadsheet object. The corresponding interfaces are `com.sun.star.sheet.XDataPilotTablesSupplier`, `com.sun.star.sheet.XScenariosSupplier`, `com.sun.star.sheet.XSheetOutline` and `com.sun.star.sheet.XSheetAuditing`.
- Cells can be inserted, and entire cell ranges can be removed, moved or copied on the spreadsheet level using `com.sun.star.sheet.XCellRangeMovement`.
- Drawing elements in a spreadsheet are part of the draw page available through `com.sun.star.drawing.XDrawPageSupplier`.
- Certain sheet printing features are accessed at the spreadsheet. The `com.sun.star.sheet.XPrintAreas` and `com.sun.star.sheet.XSheetPageBreak` are used to get page breaks and control print areas.
- The spreadsheet maintains charts. The interface `com.sun.star.table.XTableChartsSupplier` controls charts in the spreadsheet.

- All cell annotations can be retrieved on the spreadsheet level with `com.sun.star.sheet.XSheetAnnotationsSupplier`.
- A spreadsheet can be permanently protected from changes through `com.sun.star.util.XProtectable`.

Capabilities of SheetCellRange

The spreadsheet, as well as the cell ranges in a spreadsheet are `com.sun.star.sheet.SheetCellRange` services. A `SheetCellRange` is a rectangular range of calculation cells that includes the following services:

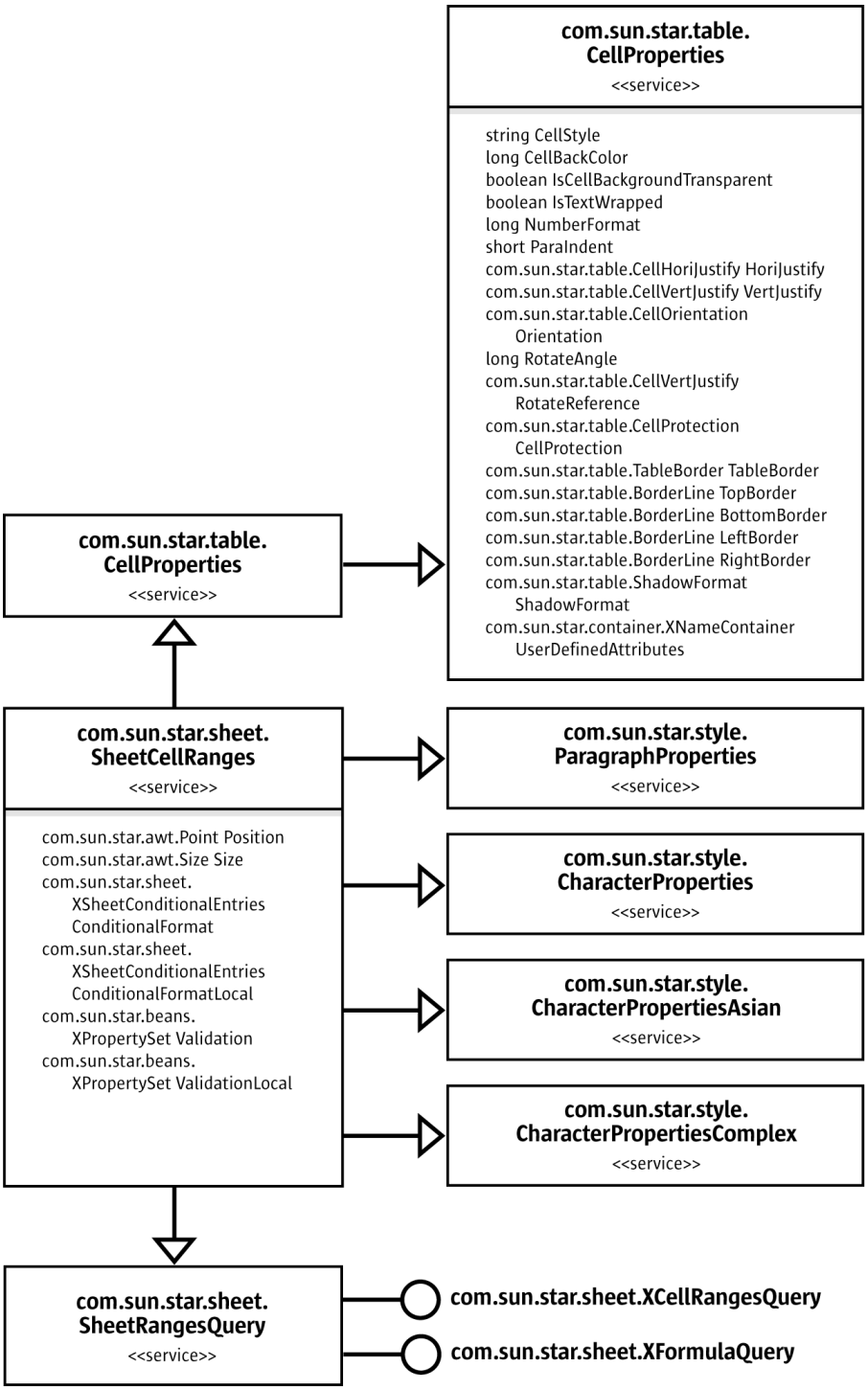


Illustration 83: Services supported by SheetCellRange

The interfaces supported by a SheetCellRange are depicted in the following illustration:

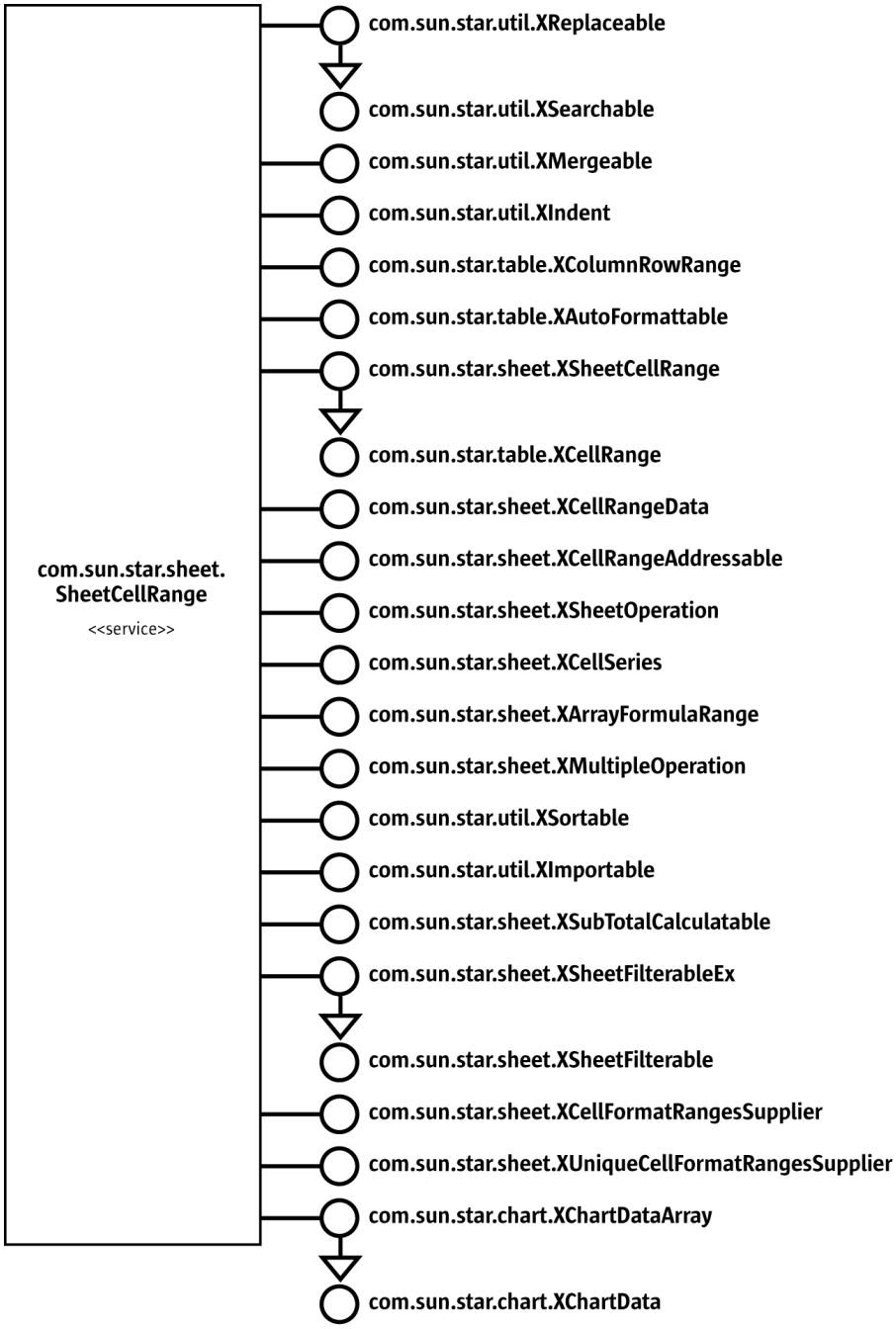


Illustration 84: SheetCellRange Interfaces

A `SheetCellRange` has the following capabilities:

- Supplies cells and sub-ranges of cells, as well as rows and columns. It has the interfaces `com.sun.star.sheet.XSheetCellRange` and `com.sun.star.table.XColumnRowRange`.
- Performs calculations with a `SheetCellRange`. The interface `com.sun.star.sheet.XSheetOperation` is for aggregate operations, `com.sun.star.sheet.XMultipleOperation` copies formulas adjusting their cell references, `com.sun.star.sheet.XSubTotalCalculatable` applies and removes sub totals, and `com.sun.star.sheet.XArrayFormulaRange` handles array formulas.

- Formats cells in a range. The settings affect all cells in the range. There are cell properties, character properties and paragraph properties for formatting purposes. Additionally, a SheetCellRange supports auto formats with `com.sun.star.table.XAutoFormattable` and the content of the cells can be indented using `com.sun.star.util.XIndent`. The interfaces `com.sun.star.sheet.XCellFormatRangesSupplier` and `com.sun.star.sheet.XUniqueCellFormatRangesSupplier` obtain enumeration of cells that differ in formatting.
- Works with the data in a cell range through a sequence of sequences of `any` that maps to the two-dimensional cell array of the range. This array is available through `com.sun.star.sheet.XCellRangeData`.
- Fills a cell range with data series automatically through its interface `com.sun.star.sheet.XCellSeries`.
- Imports data from a database using `com.sun.star.util.XImportable`.
- Searches and replaces cell contents using `com.sun.star.util.XSearchable`.
- Perform queries for cell contents, such as formula cells, formula result types, or empty cells. The interface `com.sun.star.sheet.XCellRangesQuery` of the included `com.sun.star.sheet.SheetRangesQuery` service is responsible for this task.
- Merges cells into a single cell through `com.sun.star.util.XMergeable`.
- Sorts and filters the content of a SheetCellRange, using `com.sun.star.util.XSortable`, `com.sun.star.sheet.XSheetFilterable` and `com.sun.star.sheet.XSheetFilterableEx`.
- Provides its unique range address in the spreadsheet document, that is, the start column and row, end column and row, and the sheet where it is located. The `com.sun.star.sheet.XCellRangeAddressable:getRangeAddress()` returns the corresponding address description struct `com.sun.star.table.CellRangeAddress`.
- Charts can be based on a `SheetCellRange`, because it supports `com.sun.star.chart.XChartDataArray`.

Capabilities of SheetCell

A `com.sun.star.sheet.SheetCell` is the base unit of OpenOffice.org Calc tables. Values, formulas and text required for calculation jobs are all written into sheet cells. The `SheetCell` includes the following services:

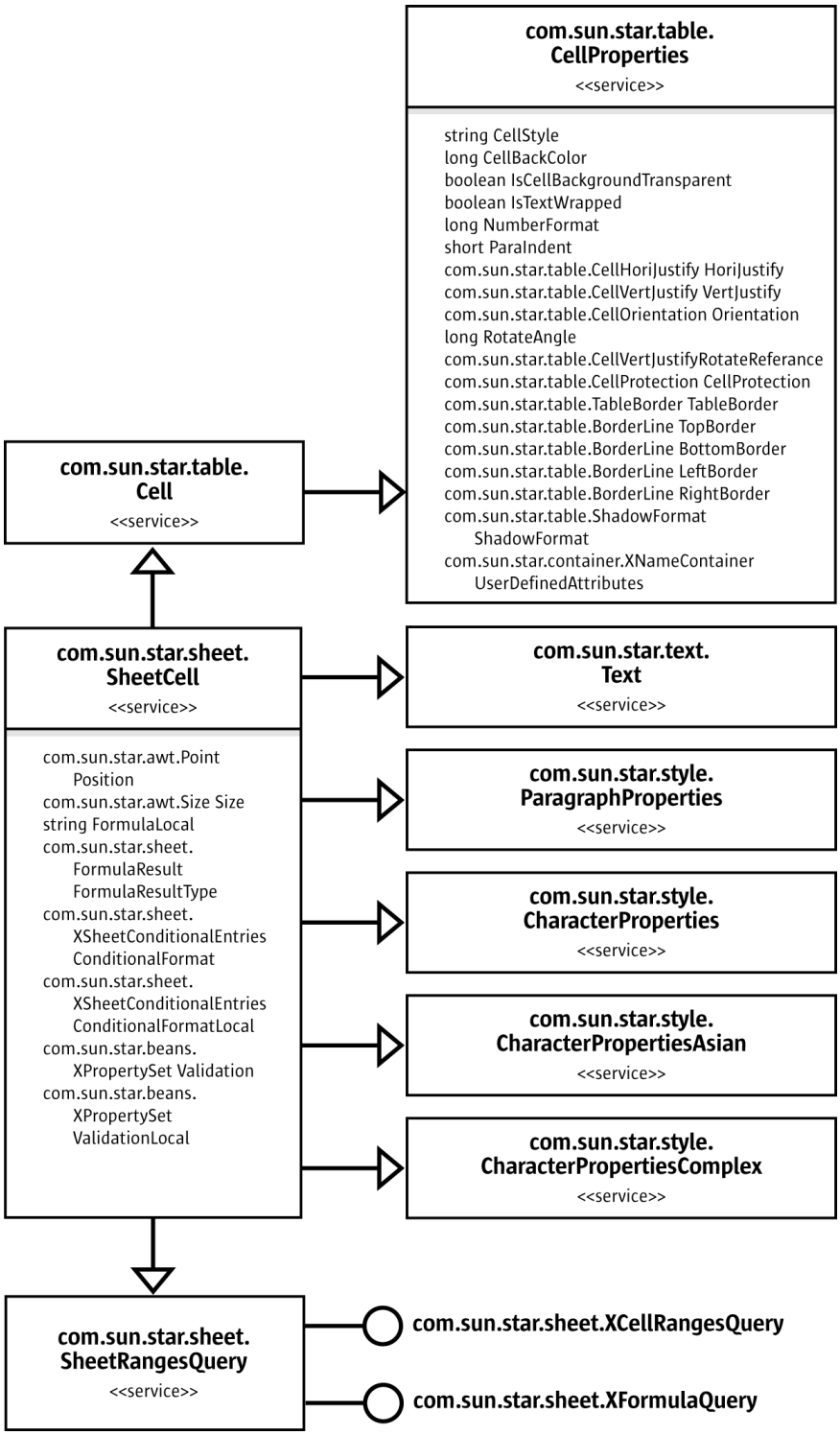


Illustration 85: SheetCell

The SheetCell exports the following interfaces:

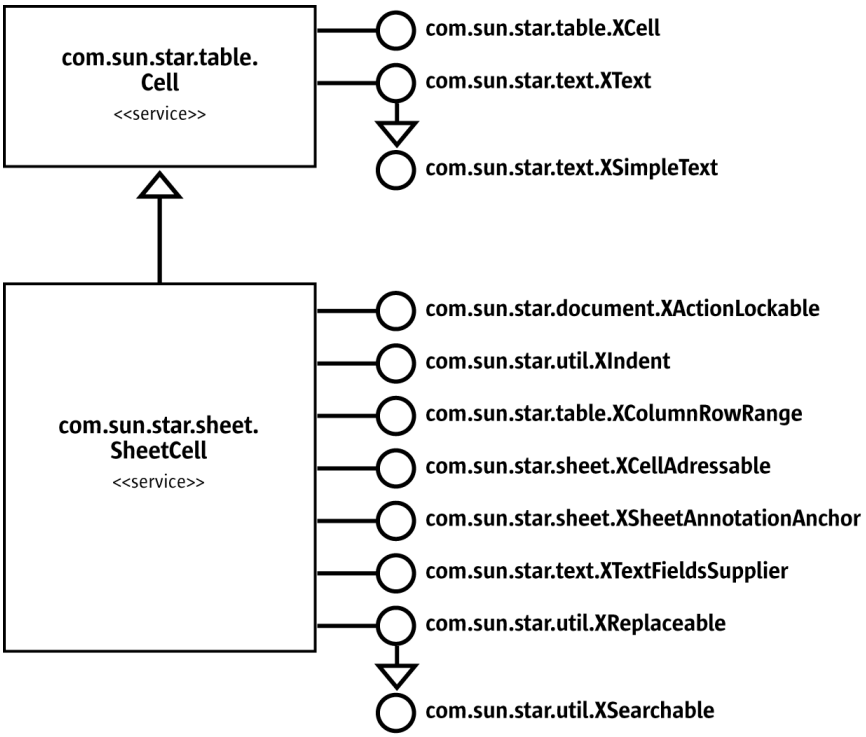


Illustration 86: SheetCell Interfaces

The SheetCell service has the following capabilities:

- It can access the cell content. It can contain numeric *values* that are used for calculations, *formulas* that operate on these values, and *text* supporting full-featured formatting and hyperlink text fields. The access to the cell values and formulas is provided through the SheetCell parent service com.sun.star.table.Cell. The interface com.sun.star.table.XCell is capable of manipulating the values and formulas in a cell. For text, the service com.sun.star.text.Text with the main interface com.sun.star.text.XText is available at a SheetCell. Its text fields are accessed through com.sun.star.text.XTextFieldsSupplier.
- A SheetCell is a special case of a SheetCellRange. As such, it has all capabilities of the com.sun.star.sheet.SheetCellRange described above.
- It can have an annotation: com.sun.star.sheet.XSheetAnnotationAnchor.
- It can provide its unique cell address in the spreadsheet document, that is, its column, row and the sheet it is located in. The com.sun.star.sheet.XCellAddressable:getCellAddress() returns the appropriate com.sun.star.table.CellAddress struct.
- It can be locked temporarily against user interaction with com.sun.star.document.XActionLockable.

Capabilities of SheetCellRanges Container

The container of com.sun.star.sheet.SheetCellRanges is used where several cell ranges have to be handled at once for cell query results and other situations. The SheetCellRanges service includes cell, paragraph and character property services, and it offers a query option:

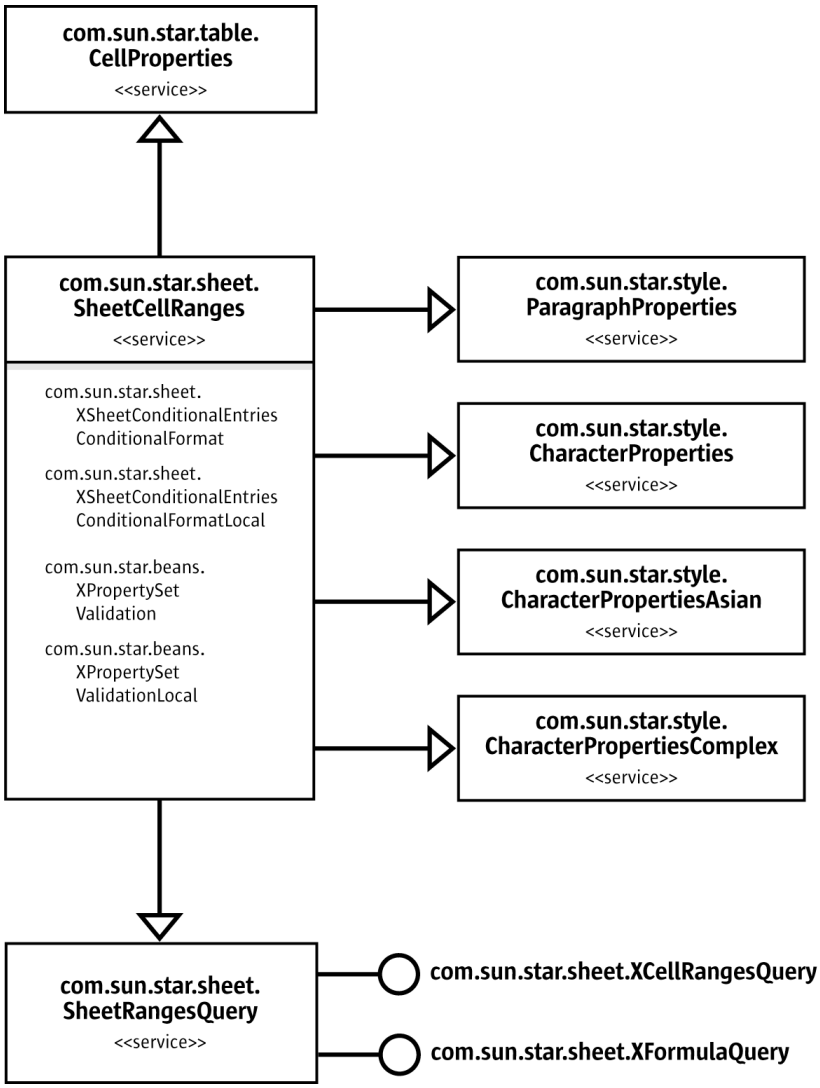


Illustration 87: Services of SheetCellRanges

The interfaces of `com.sun.star.sheet.SheetCellRanges` are element accesses for the ranges in the `SheetCellRanges` container. These interfaces are discussed below.

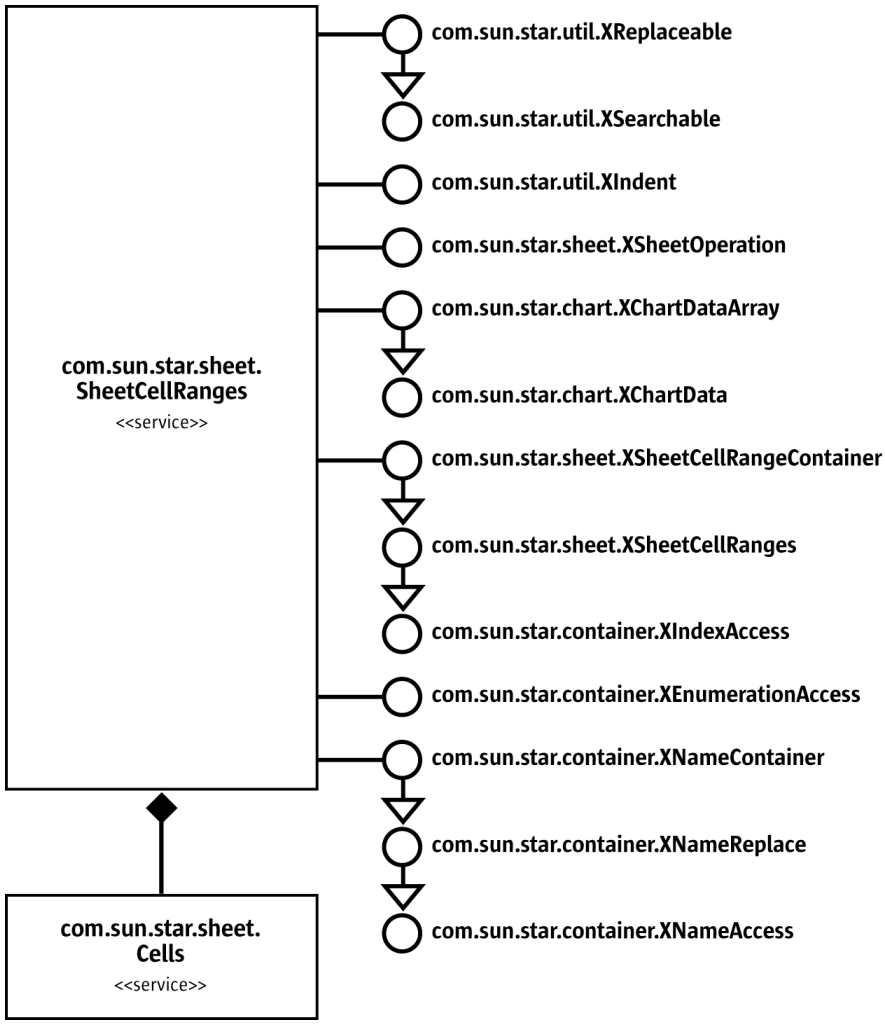


Illustration 88: Implemented interfaces of SheetCellRanges

The `SheetCellRanges` container has the following capabilities:

- It can be formatted using the character, paragraph and cell property services it includes.
- It yields independent cell ranges through the element access interfaces `com.sun.star.container.XIndexAccess`, `com.sun.star.container.XNameAccess` and `com.sun.star.container.XEnumerationAccess`.
- It can access, replace, append and remove ranges by *name* through `com.sun.star.container.XNameContainer`
- It can add new ranges to `SheetCellRanges` by their *address descriptions,* access the ranges by *index,* and obtain the *cells* in the ranges. This is possible through the interface `com.sun.star.sheet.XSheetCellRangeContainer` that was originally based on `com.sun.star.container.XIndexAccess`. The `SheetCellRanges` maintain a sub-container of all cells in the ranges that are not empty, obtainable through the `getCells()` method.
- It can enumerate the ranges using `com.sun.star.container.XEnumerationAccess`.
- It can query the ranges for certain cell contents, such as formula cells, formula result types or empty cells. The interface `com.sun.star.sheet.XCellRangesQuery` of the included `com.sun.star.sheet.SheetRangesQuery` service is responsible for this task.

- The `SheetCellRanges` supports selected `SheetCellRange` features, such as searching and replacing, indenting, sheet operations and charting.

Capabilities of Columns and Rows

All cell ranges are organized in columns and rows, therefore column and row containers are retrieved from a spreadsheet, as well as from sub-ranges of a spreadsheet through `com.sun.star.table.XColumnRowRange`. These containers are `com.sun.star.table.TableColumns` and `com.sun.star.table.TableRows`. Both containers support index and enumeration access. Only the `TableColumns` supports name access to the single columns and rows (`com.sun.star.table.TableColumn` and `com.sun.star.table.TableRow`) of a `SheetCellRange`.

The following UML charts show table columns and rows. The first chart shows columns:

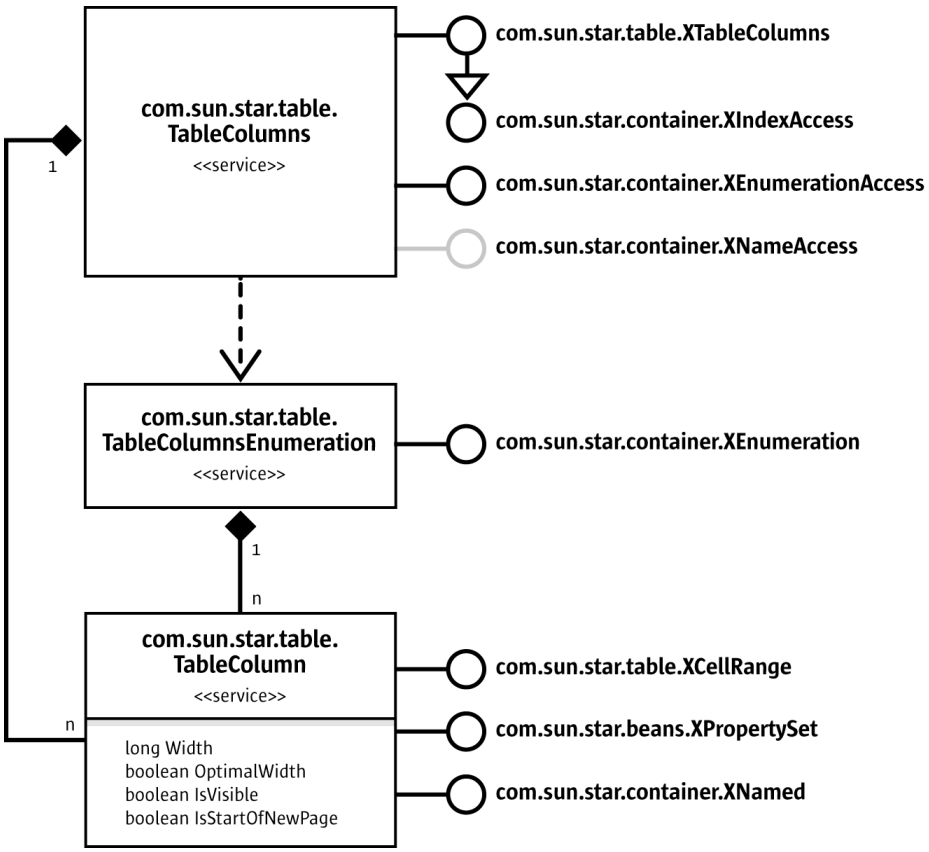


Illustration 89: Collection of table columns

The collection of table rows differs from the collection of columns, that is, it does not support `com.sun.star.container.XNameAccess`:

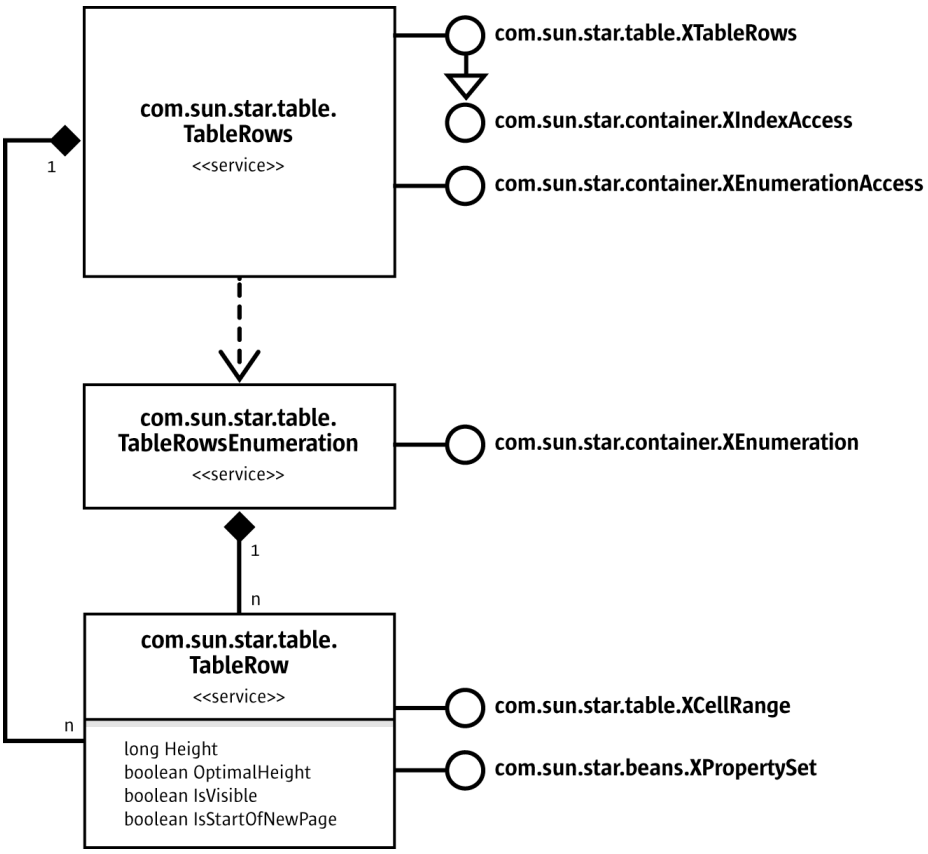


Illustration 90: Collection of table rows

The services for table rows and columns control the table structure and grid size of a cell range:

- The containers for columns and rows have methods to insert and remove columns, and rows by index in their main interfaces `com.sun.star.table.XTableRows` and `com.sun.star.table.XTableColumns`.
- The services `TableColumn` and `TableRow` have properties to adjust their column width and row height, toggle their visibility, and set page breaks.

Spreadsheet

A spreadsheet is a cell range with additional interfaces and is represented by the service `com.sun.star.sheet.Spreadsheet`.

Properties of Spreadsheet

The properties of a spreadsheet deal with its visibility and its page style:

Properties of `com.sun.star.sheet.Spreadsheet`	
`IsVisible`	boolean — Determines if the sheet is visible in the GUI.

Properties of com.sun.star.sheet.Spreadsheet	
PageStyle	Contains the name of the page style of this spreadsheet. See *8.4.1 Spreadsheet Documents - Overall Document Features - Styles* for details about styles.

Naming

The spreadsheet interface com.sun.star.container.XNamed obtains and changes the name of the spreadsheet, and uses it to get a spreadsheet from the spreadsheet collection. Refer to *8.3.1 Spreadsheet Documents - Working with Spreadsheets - Document Structure - Spreadsheet Document.*

Inserting Cells, Moving and Copying Cell Ranges

The interface com.sun.star.sheet.XCellRangeMovement of the Spreadsheet service supports *inserting* and *removing* cells from a spreadsheet, and *copying* and *moving* cell contents. When cells are copied or moved, the relative references of all formulas are updated automatically. The sheet index included in the source range addresses should be equal to the index of the sheet of this interface.

Methods of com.sun.star.sheet.XCellRangeMovement	
insertCells]()	Inserts a range of empty cells at a specific position. The direction of the insertion is determined by the parameter nMode (type com.sun.star.sheet.CellInsertMode).
removeRange()	Deletes a range of cells from the spreadsheet. The parameter nMode (type com.sun.star.sheet.CellDeleteMode) determines how remaining cells will be moved.
copyRange()	Copies the contents of a cell range to another place in the document.
IDLS:com.sun.star.sheet.XCellRangeMovement:moveRange]()	Moves the contents of a cell range to another place in the document. Deletes all contents of the source range.

The following example copies a cell range to another location in the sheet. (Spreadsheet/SpreadsheetSample.java)

```
/** Copies a cell range to another place in the sheet.
    @param xSheet  The XSpreadsheet interface of the spreadsheet.
    @param aDestCell  The address of the first cell of the destination range.
    @param aSourceRange  The source range address.
 */
public void doMovementExample(com.sun.star.sheet.XSpreadsheet xSheet,
        com.sun.star.table.CellAddress aDestCell, com.sun.star.table.CellRangeAddress aSourceRange)
        throws RuntimeException, Exception {
    com.sun.star.sheet.XCellRangeMovement xMovement = (com.sun.star.sheet.XCellRangeMovement)
        UnoRuntime.queryInterface(com.sun.star.sheet.XCellRangeMovement.class, xSheet);
    xMovement.copyRange(aDestCell, aSourceRange);
}
```

Page Breaks

The methods getColumnPageBreaks() and getRowPageBreaks() of the interface com.sun.star.sheet.XSheetPageBreak return the positions of column and row page breaks, represented by a sequence of com.sun.star.sheet.TablePageBreakData structs. Each struct contains the position of the page break and a boolean property that determines if the page break was inserted manually. Inserting and removing a manual page break uses the property IsStartOfNewPage of the services com.sun.star.table.TableColumn and com.sun.star.table.TableRow.

The following example prints the positions of all the automatic column page breaks: (Spreadsheet/SpreadsheetSample.java)

```
// --- Print automatic column page breaks ---
com.sun.star.sheet.XSheetPageBreak xPageBreak = (com.sun.star.sheet.XSheetPageBreak)
    UnoRuntime.queryInterface(com.sun.star.sheet.XSheetPageBreak.class, xSheet);
com.sun.star.sheet.TablePageBreakData[] aPageBreakArray = xPageBreak.getColumnPageBreaks();

System.out.print("Automatic column page breaks:");
for (int nIndex = 0; nIndex < aPageBreakArray.length; ++nIndex)
    if (!aPageBreakArray[nIndex].ManualBreak)
        System.out.print( " " + aPageBreakArray[nIndex].Position);
System.out.println();
```

Cell Ranges

A cell range is a rectangular range of cells. It is represented by the service `com.sun.star.sheet.SheetCellRange`.

Properties of Cell Ranges

The cell range properties deal with the position and size of a range, conditional formats, and cell validation during user input.

Properties of `com.sun.star.sheet.SheetCellRange`	
`Position` `Size`	The position and size of the cell in 100^{th} of a millimeter. The position is relative to the first cell of the spreadsheet. Note, that this is not always the first visible cell.
`ConditionalFormat` `ConditionalFormatLocal`	Used to access conditional formats. See *8.3.2 Spreadsheet Documents - Working with Spreadsheets - Formatting - Conditional Formats* for details.
`Validation` `ValidationLocal`	Used to access data validation. See *8.3.11 Spreadsheet Documents - Working with Spreadsheets - Other Table Operations - Data Validation* for details.

This service extends the service `com.sun.star.table.CellRange` to provide common table cell range functionality.

Cell and Cell Range Access

The interface `com.sun.star.sheet.XSheetCellRange` is derived from `com.sun.star.table.XCellRange`. It provides access to cells of the range and sub ranges, and is supported by the spreadsheet and sub-ranges of a spreadsheet. The methods in `com.sun.star.sheet.XSheetCellRange` are:

```
com::sun::star::table::XCell getCellByPosition( [in] long nColumn, [in] long nRow)
com::sun::star::table::XCellRange getCellRangeByPosition( [in] long nLeft, [in] long nTop,
                                                          [in] long nRight, [in] long nBottom)
com::sun::star::table::XCellRange getCellRangeByName ( [in] string aRange)
com::sun::star::sheet::XSpreadsheet getSpreadsheet()
```

The interface `com.sun.star.table.XCellRange` provides methods to access cell ranges and single cells from a cell range.

Cells are retrieved by their position. Cell addresses consist of a row index and a column index. The index is zero-based, that is, the index 0 means the first row or column of the table.

Cell ranges are retrieved:

by position
Addresses of cell ranges consist of indexes to the first and last row, and the first and last column. Range indexes are always zero-based, that is, the index 0 points to the first row or column of the table.

by name
It is possible to address a cell range over its name in A1:B2 notation as it would appear in the application.



In a spreadsheet, “A1:B2”, “$C$1:$D$2”, or “E5” are valid ranges. Even user defined cell names, range names, or database range names can be used.

Additionally, `XCellRange` contains the method `getSpreadsheet()` that returns the `com.sun.star.sheet.XSpreadsheet` interface of the spreadsheet which contains the cell range.

```
// --- First cell in a cell range. ---
com.sun.star.table.XCell xCell = xCellRange.getCellByPosition(0, 0);

// --- Spreadsheet that contains the cell range. ---
com.sun.star.sheet.XSpreadsheet xSheet = xCellRange.getSpreadsheet();
```

There are no methods to modify the contents of *all* cells of a cell range. Access to cell range formatting is supported. Refer to the chapter *8.3.2 Spreadsheet Documents - Working with Spreadsheets - Formatting* for additional details.

In the following example, `xRange` is an existing cell range (a `com.sun.star.table.XCellRange` interface): (Spreadsheet/GeneralTableSample.java)

```
com.sun.star.beans.XPropertySet xPropSet = null;
com.sun.star.table.XCellRange xCellRange = null;

// *** Accessing a CELL RANGE ***

// Accessing a cell range over its position.
xCellRange = xRange.getCellRangeByPosition(2, 0, 3, 1);

// Change properties of the range.
xPropSet = (com.sun.star.beans.XPropertySet)
    UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, xCellRange);
xPropSet.setPropertyValue("CellBackColor", new Integer(0x8080FF));

// Accessing a cell range over its name.
xCellRange = xRange.getCellRangeByName("C4:D5");

// Change properties of the range.
xPropSet = (com.sun.star.beans.XPropertySet)
    UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, xCellRange);
xPropSet.setPropertyValue("CellBackColor", new Integer(0xFFFF80));
```

Merging Cell Ranges into a Single Cell

The cell range interface `com.sun.star.util.XMergeable` merges and undoes merged cell ranges.

- The method `merge()` merges or undoes merged the whole cell range.
- The method `getIsMerged()` determines if the cell range is completely merged.

(Spreadsheet/SpreadsheetSample.java)

```
// --- Merge cells. ---
com.sun.star.util.XMergeable xMerge = (com.sun.star.util.XMergeable)
    UnoRuntime.queryInterface(com.sun.star.util.XMergeable.class, xCellRange);
xMerge.merge(true);
```

Column and Row Access

The cell range interface `com.sun.star.table.XColumnRowRange` accesses the column and row ranges in the current cell range. A column or row range contains all the cells in the selected column or row. This type of range has additional properties, such as, visibility, and width or

height. For more information, see *8.3.1 Spreadsheet Documents - Working with Spreadsheets - Document Structure - Columns and Rows.*

- The method `getColumns()` returns the interface `com.sun.star.table.XTableColumns` of the collection of columns.
- The method `getRows()` returns the interface `com.sun.star.table.XTableRows` of the collection of rows.

(Spreadsheet/SpreadsheetSample.java)

```
// --- Column properties. ---
com.sun.star.table.XColumnRowRange xColRowRange = (com.sun.star.table.XColumnRowRange)
    UnoRuntime.queryInterface(com.sun.star.table.XColumnRowRange.class, xCellRange);
com.sun.star.table.XTableColumns xColumns = xColRowRange.getColumns();

Object aColumnObj = xColumns.getByIndex(0);
xPropSet = (com.sun.star.beans.XPropertySet) UnoRuntime.queryInterface(
    com.sun.star.beans.XPropertySet.class, aColumnObj);
xPropSet.setPropertyValue( "Width", new Integer( 6000 ) );

com.sun.star.container.XNamed xNamed = (com.sun.star.container.XNamed)
    UnoRuntime.queryInterface(com.sun.star.container.XNamed.class, aColumnObj);
System.out.println("The name of the wide column is " + xNamed.getName() + ".");
```

Data Array

The contents of a cell range that are stored in a 2-dimensional array of objects are set and obtained by the interface `com.sun.star.sheet.XCellRangeData`.

- The method `getDataArray()` returns a 2-dimensional array with the contents of all cells of the range.
- The method `setDataArray()` fills the data of the passed array into the cells. An empty cell is created by an empty string. The size of the array has to fit in the size of the cell range.

The following example uses the cell range `xCellRange` that has the size of 2 columns and 3 rows. (Spreadsheet/SpreadsheetSample.java)

```
// --- Cell range data ---
com.sun.star.sheet.XCellRangeData xData = (com.sun.star.sheet.XCellRangeData)
    UnoRuntime.queryInterface(com.sun.star.sheet.XCellRangeData.class, xCellRange);

Object[][] aValues =
{
    {new Double(1.1), new Integer(10)},
    {new Double(2.2), new String("")},
    {new Double(3.3), new String("Text")}
};

xData.setDataArray(aValues);
```

Absolute Address

The method `getCellRangeAddress()` of the interface `com.sun.star.sheet.XCellRangeAddressable` returns a `com.sun.star.table.CellRangeAddress` struct that contains the absolute address of the cell in the spreadsheet document, including the sheet index. This is useful to get the address of cell ranges returned by other methods. (Spreadsheet/SpreadsheetSample.java)

```
// --- Get cell range address. ---
com.sun.star.sheet.XCellRangeAddressable xRangeAddr = (com.sun.star.sheet.XCellRangeAddressable)
    UnoRuntime.queryInterface(com.sun.star.sheet.XCellRangeAddressable.class, xCellRange);
aRangeAddress = xRangeAddr.getRangeAddress();
System.out.println("Address of this range:  Sheet=" + aRangeAddress.Sheet);
System.out.println(
    "Start column=" + aRangeAddress.StartColumn + ";  Start row=" + aRangeAddress.StartRow);
System.out.println(
    "End column  =" + aRangeAddress.EndColumn   + ";  End row  =" + aRangeAddress.EndRow);
```

Fill Series

The interface `com.sun.star.sheet.XCellSeries` fills out each cell of a cell range with values based on a start value, step count and fill mode. It is possible to fill a series in each direction, specified by a `com.sun.star.sheet.FillDirection` constant. If the fill direction is horizontal, each row of the cell range forms a separate series. Similarly each column forms a series on a vertical fill.

- The method `fillSeries()` uses the first cell of each series as start value. For example, if the fill direction is "To top", the bottom-most cell of each column is used as the start value. It expects a fill mode to be used to continue the start value, a `com.sun.star.sheet.FillMode` constant. If the values are dates, `com.sun.star.sheet.FillDateMode` constants describes the mode how the dates are calculated. If the series reaches the specified end value, the calculation is stopped.
- The method `fillAuto()` determines the fill mode and step count automatically. It takes a parameter containing the number of cells to be examined. For example, if the fill direction is "To top" and the specified number of cells is three, the three bottom-most cells of each column are used to continue the series.

The following example may operate on the following spreadsheet:

	A	B	C	D	E	F	G
1	1						
2	4						
3	01/30/2002						
4					Text 10		
5	Jan						10
6							
7	1	2					
8	05/28/2002	02/28/2002					
9	6	4					

Inserting filled series in Java: (Spreadsheet/SpreadsheetSample.java)

```
public void doSeriesSample(com.sun.star.sheet.XSpreadsheet xSheet) {
    com.sun.star.sheet.XCellSeries xSeries = null;

    // Fill 2 rows linear with end value -> 2nd series is not filled completely
    xSeries = getCellSeries(xSheet, "A1:E2");
    xSeries.fillSeries(
        com.sun.star.sheet.FillDirection.TO_RIGHT, com.sun.star.sheet.FillMode.LINEAR,
        com.sun.star.sheet.FillDateMode.FILL_DATE_DAY, 2, 9);

    // Add months to a date
    xSeries = getCellSeries(xSheet, "A3:E3");
    xSeries.fillSeries(
        com.sun.star.sheet.FillDirection.TO_RIGHT, com.sun.star.sheet.FillMode.DATE,
        com.sun.star.sheet.FillDateMode.FILL_DATE_MONTH, 1, 0x7FFFFFFF);

    // Fill right to left with a text containing a value
    xSeries = getCellSeries(xSheet, "A4:E4");
    xSeries.fillSeries(
        com.sun.star.sheet.FillDirection.TO_LEFT, com.sun.star.sheet.FillMode.LINEAR,
        com.sun.star.sheet.FillDateMode.FILL_DATE_DAY, 10, 0x7FFFFFFF);

    // Fill with an user defined list
    xSeries = getCellSeries(xSheet, "A5:E5");
    xSeries.fillSeries(
        com.sun.star.sheet.FillDirection.TO_RIGHT, com.sun.star.sheet.FillMode.AUTO,
        com.sun.star.sheet.FillDateMode.FILL_DATE_DAY, 1, 0x7FFFFFFF);

    // Fill bottom to top with a geometric series
    xSeries = getCellSeries(xSheet, "G1:G5");
    xSeries.fillSeries(
        com.sun.star.sheet.FillDirection.TO_TOP, com.sun.star.sheet.FillMode.GROWTH,
        com.sun.star.sheet.FillDateMode.FILL_DATE_DAY, 2, 0x7FFFFFFF);

    // Auto fill
    xSeries = getCellSeries(xSheet, "A7:G9");
    xSeries.fillAuto(com.sun.star.sheet.FillDirection.TO_RIGHT, 2);
}


/** Returns the XCellSeries interface of a cell range.
    @param xSheet  The spreadsheet containing the cell range.
    @param aRange  The address of the cell range.
    @return  The XCellSeries interface. */
private com.sun.star.sheet.XCellSeries getCellSeries(
        com.sun.star.sheet.XSpreadsheet xSheet, String aRange) {
    return (com.sun.star.sheet.XCellSeries) UnoRuntime.queryInterface(
        com.sun.star.sheet.XCellSeries.class, xSheet.getCellRangeByName(aRange));
}
```

This example produces the following result:

	A	B	C	D	E	F	G
1	1	3	5	7	9		160
2	4	6	8				80
3	01/30/2002	02/28/2002	03/30/2002	04/30/2002	05/30/2002		40
4	Text 50	Text 40	Text 30	Text 20	Text 10		20
5	Jan	Feb	Mar	Apr	May		10
6							
7	1	2	3	4	5	6	7
8	05/28/2002	02/28/2002	11/28/2001	08/28/2001	05/28/2001	02/28/2001	11/28/2000
9	6	4	2	0	-2	-4	-6

Operations

The cell range interface `com.sun.star.sheet.XSheetOperation` computes a value based on the contents of all cells of a cell range or clears specific contents of the cells.

- The method `computeFunction()` returns the result of the calculation. The constants `com.sun.star.sheet.GeneralFunction` specify the calculation method.

- The method `clearContents()` clears contents of the cells used. The parameter describes the contents to clear, using the constants of `com.sun.star.sheet.CellFlags`.

The following code shows how to compute the average of a cell range and clear the cell contents:

```
// --- Sheet operation. ---
// Compute a function
com.sun.star.sheet.XSheetOperation xSheetOp = (com.sun.star.sheet.XSheetOperation)
    UnoRuntime.queryInterface(com.sun.star.sheet.XSheetOperation.class, xCellRange);

double fResult = xSheetOp.computeFunction(com.sun.star.sheet.GeneralFunction.AVERAGE);
System.out.println("Average value of the data table A10:C30: " + fResult);

// Clear cell contents
xSheetOp.clearContents(
    com.sun.star.sheet.CellFlags.ANNOTATION | com.sun.star.sheet.CellFlags.OBJECTS);
```

Multiple Operations

A multiple operation combines a series of formulas with a variable and a series of values. The results of each formula with each value is shown in the table. Additionally, it is possible to calculate a single formula with two variables using a 2-value series. The method `setTableOperation()` of the interface `com.sun.star.sheet.XMultipleOperation` inserts a multiple operation range.

The following example shows how to calculate the values 1 to 5 raised to the powers of 1 to 5 (each value to each power). The first column contains the base values, and the first row the exponents, for example, cell E3 contains the result of 2^4. Below there are three trigonometrical functions calculated based on a series of values, for example, cell C11 contains the result of cos(0.2).

	A	B	C	D	E	F	G
1	=A2^B1	1	2	3	4	5	
2	1						
3	2						
4	3						
5	4						
6	5						
7							
8		=SIN(A8)	=COS(A8)	=TAN(A8)			
9	0						
10	0.1						
11	0.2						
12	0.3						
13	0.4						

Note that the value series have to be included in the multiple operations cell range, but not the formula cell range (in the second example). The references in the formulas address any cell outside of the area to be filled. The column cell and row cell parameter have to reference these cells exactly. In the second example, a row cell address does not have to be used, because the row contains the formulas. (Spreadsheet/SpreadsheetSample.java)

```
public void InsertMultipleOperation(com.sun.star.sheet.XSpreadsheet xSheet)
        throws RuntimeException, Exception {
    // --- Two independent value series ---
    com.sun.star.table.CellRangeAddress aFormulaRange = createCellRangeAddress(xSheet, "A1");
    com.sun.star.table.CellAddress aColCell = createCellAddress(xSheet, "A2");
    com.sun.star.table.CellAddress aRowCell = createCellAddress(xSheet, "B1");

    com.sun.star.table.XCellRange xCellRange = xSheet.getCellRangeByName("A1:F6");
    com.sun.star.sheet.XMultipleOperation xMultOp = (com.sun.star.sheet.XMultipleOperation)
        UnoRuntime.queryInterface(com.sun.star.sheet.XMultipleOperation.class, xCellRange);
    xMultOp.setTableOperation(
        aFormulaRange, com.sun.star.sheet.TableOperationMode.BOTH, aColCell, aRowCell);


    // --- A value series, a formula series ---
    aFormulaRange = createCellRangeAddress(xSheet, "B8:D8");
    aColCell = createCellAddress(xSheet, "A8");
    // Row cell not needed

    xCellRange = xSheet.getCellRangeByName("A9:D13");
    xMultOp = (com.sun.star.sheet.XMultipleOperation)
        UnoRuntime.queryInterface(com.sun.star.sheet.XMultipleOperation.class, xCellRange);
    xMultOp.setTableOperation(
        aFormulaRange, com.sun.star.sheet.TableOperationMode.COLUMN, aColCell, aRowCell);
}


/** Creates a com.sun.star.table.CellAddress and initializes it
    with the given range.
    @param xSheet  The XSpreadsheet interface of the spreadsheet.
    @param aCell  The address of the cell (or a named cell).
 */
public com.sun.star.table.CellAddress createCellAddress(
        com.sun.star.sheet.XSpreadsheet xSheet,
        String aCell ) throws RuntimeException, Exception {
    com.sun.star.sheet.XCellAddressable xAddr = (com.sun.star.sheet.XCellAddressable)
        UnoRuntime.queryInterface(com.sun.star.sheet.XCellAddressable.class,
            xSheet.getCellRangeByName(aCell).getCellByPosition(0, 0));
    return xAddr.getCellAddress();
}


/** Creates a com.sun.star.table.CellRangeAddress and initializes
    it with the given range.
    @param xSheet  The XSpreadsheet interface of the spreadsheet.
    @param aRange  The address of the cell range (or a named range).
 */
public com.sun.star.table.CellRangeAddress createCellRangeAddress(
        com.sun.star.sheet.XSpreadsheet xSheet, String aRange) {
    com.sun.star.sheet.XCellRangeAddressable xAddr = (com.sun.star.sheet.XCellRangeAddressable)
        UnoRuntime.queryInterface(com.sun.star.sheet.XCellRangeAddressable.class,
            xSheet.getCellRangeByName(aRange));
    return xAddr.getRangeAddress();
}
```

Handling Array Formulas

The interface `com.sun.star.sheet.XArrayFormulaRange` handles array formulas.

- If the whole cell range contains an array formula, the method `getArrayFormula()` returns the formula string, otherwise an empty string is returned.
- The method `setArrayFormula()` sets an array formula to the complete cell range.

(Spreadsheet/SpreadsheetSample.java)

```
    // --- Array formulas ---
    com.sun.star.sheet.XArrayFormulaRange xArrayFormula = (com.sun.star.sheet.XArrayFormulaRange)
        UnoRuntime.queryInterface(com.sun.star.sheet.XArrayFormulaRange.class, xCellRange);
    // Insert a 3x3 unit matrix.
    xArrayFormula.setArrayFormula("=A10:C12");
    System.out.println("Array formula is: " + xArrayFormula.getArrayFormula());
```



Due to a bug, this interface does not work correctly in the current implementation. The method accepts the translated function names, but not the English names. This is inconsistent to the method `setFormula()` of the interface `com.sun.star.table.XCell`.

Cells

A single cell of a spreadsheet is represented by the service `com.sun.star.sheet.SheetCell`. This service extends the service `com.sun.star.table.Cell`, that provides fundamental table cell functionality, such as setting formulas, values and text of a cell.

Properties of SheetCell

The service `com.sun.star.sheet.SheetCell` introduces new properties and interfaces, extending the formatting-related cell properties of `com.sun.star.table.Cell`.

Properties of `com.sun.star.sheet.SheetCell`	
`Position` `Size`	The position and size of the cell in 100th of a millimeter. The position is relative to the first cell of the spreadsheet. Note that this is not always the first visible cell.
`FormulaLocal`	Used to query or set a formula using function names of the current language.
`FormulaResultType`	The type of the result. It is a constant from the set `com.sun.star.sheet.FormulaResult`.
`IDLS:com.sun.star.sheet.SheetCell:Conditional-Format]` `ConditionalFormatLocal`	Used to access conditional formats. See *8.3.2 Spreadsheet Documents - Working with Spreadsheets - Formatting - Conditional Formats* for details.
`Validation` `ValidationLocal`	Used to access data validation. See *8.3.11 Spreadsheet Documents - Working with Spreadsheets - Other Table Operations - Data Validation* for details.

Access to Formulas, Values and Errors

The cell interface `com.sun.star.table.XCell` provides methods to access the value, formula, content type, and error code of a single cell:

```
void setValue( [in] double nValue)
double getValue()
void setFormula( [in] string aFormula)
string getFormula()
com::sun::star::table::CellContentType getType()
long getError()
```

The value of a cell is a floating-point number. To set a formula to a cell, the whole formula string has to be passed including the leading equality sign. The function names must be in English.



It is possible to set simple strings or even values with special number formats. In this case, the formula string consists only of a string constant or of the number as it would be entered in the table (for instance date, time, or currency values).

The method `getType()` returns a value of the enumeration `com.sun.star.table.CellContentType` indicating the type of the cell content.

The following code fragment shows how to access and modify the content, and formatting of single cells. The `xRange` is an existing cell range (a `com.sun.star.table.XCellRange` interface, described in *8.3.1 Spreadsheet Documents - Working with Spreadsheets - Document Structure - Cell Ranges*). The method `getCellByPosition()` is provided by this interface. (Spreadsheet/GeneralTableSample.java)

```
com.sun.star.beans.XPropertySet xPropSet = null;
com.sun.star.table.XCell xCell = null;

// *** Access and modify a VALUE CELL ***
xCell = xRange.getCellByPosition(0, 0);
// Set cell value.
xCell.setValue(1234);

// Get cell value.
double nDblValue = xCell.getValue() * 2;
xRange.getCellByPosition(0, 1).setValue(nDblValue);

// *** Create a FORMULA CELL and query error type ***
xCell = xRange.getCellByPosition(0, 2);
// Set formula string.
xCell.setFormula("=1/0");

// Get error type.
boolean bValid = (xCell.getError() == 0);
// Get formula string.
String aText = "The formula " + xCell.getFormula() + " is ";
aText += bValid ? "valid." : "erroneous.";

// *** Insert a TEXT CELL using the XText interface ***
xCell = xRange.getCellByPosition( 0, 3 );
com.sun.star.text.XText xCellText = (com.sun.star.text.XText)
    UnoRuntime.queryInterface( com.sun.star.text.XText.class, xCell );
com.sun.star.text.XTextCursor xTextCursor = xCellText.createTextCursor();
xCellText.insertString( xTextCursor, aText, false );

// *** Change cell properties ***
int nValue = bValid ? 0x00FF00 : 0xFF4040;
xPropSet = (com.sun.star.beans.XPropertySet) UnoRuntime.queryInterface(
    com.sun.star.beans.XPropertySet.class, xCell);
xPropSet.setPropertyValue("CellBackColor", new Integer(nValue));
```

Access to Text Content

The service `com.sun.star.text.Text` supports the modification of simple or formatted text contents. Changing text contents and text formatting is provided by the interface `com.sun.star.text.XText` as discussed in *2 First Steps*. Refer to chapter *7.3.1 Text Documents - Working with Text Documents - Word Processing - Editing Text* for further information. It implements the interfaces `com.sun.star.container.XEnumerationAccess` that provides access to the paragraphs of the text and the interface `com.sun.star.text.XText` to insert and modify text contents. For detailed information about text handling, see *7.3.1 Text Documents - Working with Text Documents - Word Processing - Editing Text*. (Spreadsheet/SpreadsheetSample.java)

```
// --- Insert two text paragraphs into the cell. ---
com.sun.star.text.XText xText = (com.sun.star.text.XText)
    UnoRuntime.queryInterface(com.sun.star.text.XText.class, xCell);
com.sun.star.text.XTextCursor xTextCursor = xText.createTextCursor();

xText.insertString(xTextCursor, "Text in first line.", false);
xText.insertControlCharacter(xTextCursor,
    com.sun.star.text.ControlCharacter.PARAGRAPH_BREAK, false);
xText.insertString(xTextCursor, "Some more text.", false);

// --- Query the separate paragraphs. ---
String aText;
com.sun.star.container.XEnumerationAccess xParaEA =
    (com.sun.star.container.XEnumerationAccess) UnoRuntime.queryInterface(
        com.sun.star.container.XEnumerationAccess.class, xCell);
com.sun.star.container.XEnumeration xParaEnum = xParaEA.createEnumeration();

// Go through the paragraphs
while (xParaEnum.hasMoreElements()) {
    Object aPortionObj = xParaEnum.nextElement();
    com.sun.star.container.XEnumerationAccess xPortionEA =
        (com.sun.star.container.XEnumerationAccess) UnoRuntime.queryInterface(
            com.sun.star.container.XEnumerationAccess.class, aPortionObj);
    com.sun.star.container.XEnumeration xPortionEnum = xPortionEA.createEnumeration();
    aText = "";

    // Go through all text portions of a paragraph and construct string.
    while (xPortionEnum.hasMoreElements()) {
        com.sun.star.text.XTextRange xRange =
            (com.sun.star.text.XTextRange) xPortionEnum.nextElement();
        aText += xRange.getString();
    }
    System.out.println("Paragraph text: " + aText);
}
```

The `SheetCell` interface `com.sun.star.text.XTextFieldsSupplier` contains methods that provide access to the collection of text fields in the cell. For details on inserting text fields, refer to *7.3.5 Text Documents - Working with Text Documents - Text Fields*.



Currently, the only possible text field in Calc cells is the hyperlink field `com.sun.star.text.textfield.URL`.

Absolute Address

The method `getCellAddress()` of the interface `com.sun.star.sheet.XCellAddressable` returns a `com.sun.star.table.CellAddress` struct that contains the absolute address of the cell in the spreadsheet document, including the sheet index. This is useful to get the address of cells returned by other methods. (Spreadsheet/SpreadsheetSample.java)

```
// --- Get cell address. ---
com.sun.star.sheet.XCellAddressable xCellAddr = (com.sun.star.sheet.XCellAddressable)
    UnoRuntime.queryInterface(com.sun.star.sheet.XCellAddressable.class, xCell);
com.sun.star.table.CellAddress aAddress = xCellAddr.getCellAddress();

String aText = "Address of this cell:  Column=" + aAddress.Column;
aText += ";  Row=" + aAddress.Row;
aText += ";  Sheet=" + aAddress.Sheet;
System.out.println(aText);
```

Cell Annotations

A spreadsheet cell may contain one annotation that consists of simple unformatted Text.

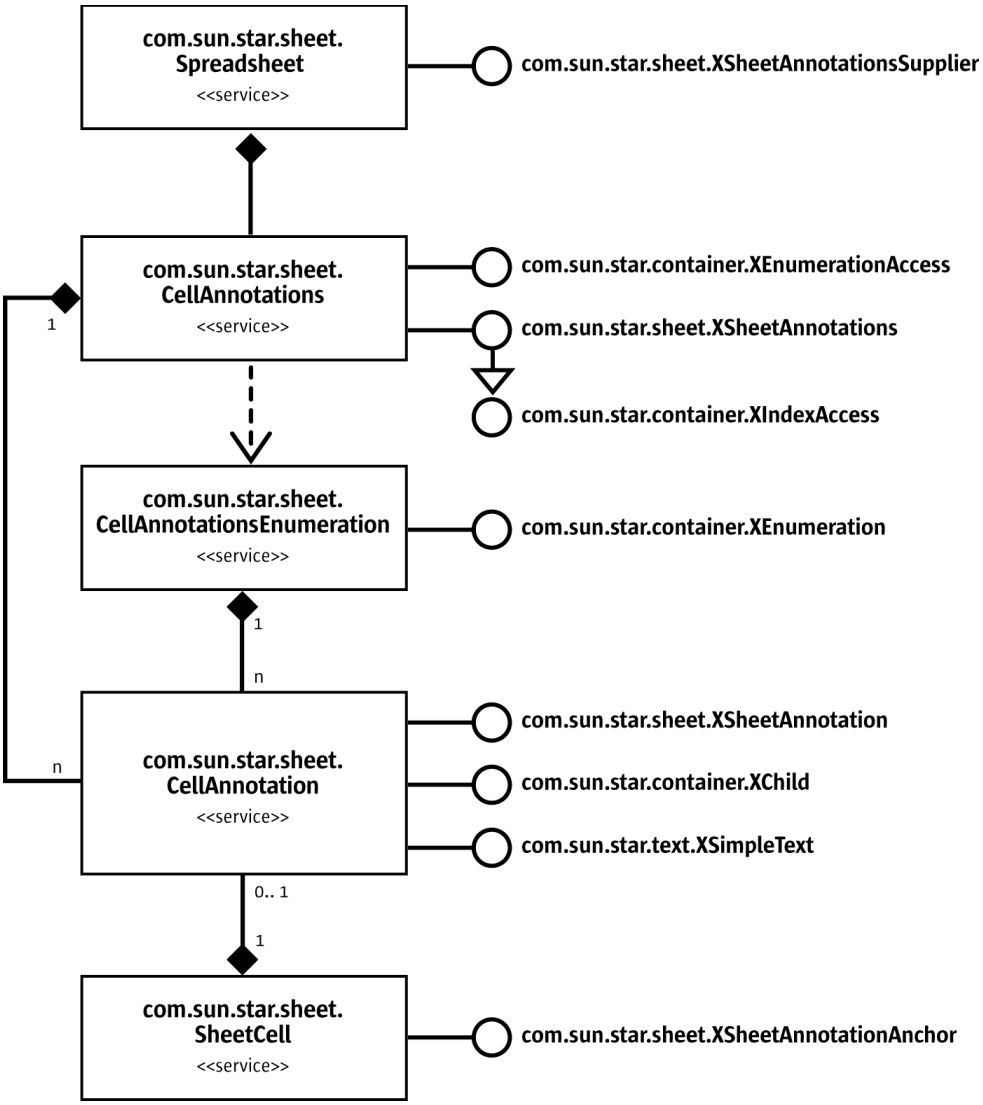


Illustration 91: Cell annotations

This service `com.sun.star.sheet.CellAnnotation` represents an annotation. It implements interfaces to manipulate the contents and access the source cell.

- The interface `com.sun.star.sheet.XSheetAnnotation` implements methods to query data of the annotation and to show and hide it. This interface is returned by the method `getAnnotation()` of the interface `com.sun.star.sheet.XSheetAnnotationAnchor`.
- The method `getParent()` of the interface `com.sun.star.container.XChild` returns the cell object that contains the annotation.
- The interface `com.sun.star.text.XSimpleText` modifies the text contents of the annotation. See *7.3.1 Text Documents - Working with Text Documents - Word Processing - Editing Text* for details.

It is possible to access the annotations through a container object from the spreadsheet or directly from a cell object.

- The method `getAnnotations()` of the interface `com.sun.star.sheet.XSheetAnnotationsSupplier` returns the interface `com.sun.star.sheet.XSheetAnnotations` of the annotations collection of this spreadsheet.

- The method `getAnnotation()` of the interface `com.sun.star.sheet.XSheetAnnotationAnchor` returns the interface `com.sun.star.sheet.XSheetAnnotation` of an annotation object.

The service `com.sun.star.sheet.CellAnnotations` represents the collection of annotations for the spreadsheet and implements two interfaces to access the annotations.

- The interface `com.sun.star.sheet.XSheetAnnotations` is derived from `com.sun.star.container.XIndexAccess` to access and remove annotations through their index. The method `insertNew()` attaches a new annotation to a cell.
- The method `createEnumeration()` of the interface `com.sun.star.container.XEnumerationAccess` creates an enumeration object, represented by the service `com.sun.star.sheet.CellAnnotationsEnumeration`, to access the annotations sequentially.

The following example inserts an annotation and makes it permanently visible. (Spreadsheet/SpreadsheetSample.java)

```
public void doAnnotationSample(
        com.sun.star.sheet.XSpreadsheet xSheet,
        int nColumn, int nRow ) throws RuntimeException, Exception {
    // create the CellAddress struct
    com.sun.star.table.XCell xCell = xSheet.getCellByPosition(nColumn, nRow);
    com.sun.star.sheet.XCellAddressable xCellAddr = (com.sun.star.sheet.XCellAddressable)
        UnoRuntime.queryInterface(com.sun.star.sheet.XCellAddressable.class, xCell);
    com.sun.star.table.CellAddress aAddress = xCellAddr.getCellAddress();

    // insert an annotation
    com.sun.star.sheet.XSheetAnnotationsSupplier xAnnotationsSupp =
        (com.sun.star.sheet.XSheetAnnotationsSupplier) UnoRuntime.queryInterface(
            com.sun.star.sheet.XSheetAnnotationsSupplier.class, xSheet);
    com.sun.star.sheet.XSheetAnnotations xAnnotations = xAnnotationsSupp.getAnnotations();
    xAnnotations.insertNew(aAddress, "This is an annotation");

    // make the annotation visible
    com.sun.star.sheet.XSheetAnnotationAnchor xAnnotAnchor =
        (com.sun.star.sheet.XSheetAnnotationAnchor) UnoRuntime.queryInterface(
            com.sun.star.sheet.XSheetAnnotationAnchor.class, xCell);
    com.sun.star.sheet.XSheetAnnotation xAnnotation = xAnnotAnchor.getAnnotation();
    xAnnotation.setIsVisible(true);
}
```

Cell Ranges and Cells Container

Cell range collections are represented by the service `com.sun.star.sheet.SheetCellRanges`. They are returned by several methods, for instance the cell query methods of `com.sun.star.sheet.SheetRangesQuery`. Besides standard container operations, it performs a few spreadsheet functions also usable with a single cell range.

Properties of SheetCellRanges

Properties of `com.sun.star.sheet.SheetCellRanges`	
`ConditionalFormat` `ConditionalFormatLocal`	Used to access conditional formats. See *8.3.2 Spreadsheet Documents - Working with Spreadsheets - Formatting - Conditional Formats* for details.
`Validation` `ValidationLocal`	Used to access data validation. See *8.3.11 Spreadsheet Documents - Working with Spreadsheets - Other Table Operations - Data Validation* for details.

Access to Single Cell Ranges in SheetCellRanges Container

The interfaces `com.sun.star.container.XEnumerationAccess` and `com.sun.star.container.XIndexAccess` iterates over all contained cell ranges by index or enumeration. With the `com.sun.star.container.XNameContainer`, it is possible to insert ranges with a user-defined name. Later the range can be found, replaced or removed using the name.

The following interfaces and service perform cell range actions on all ranges contained in the collection:

- Interface `com.sun.star.util.XReplaceable` (see *8.3.3 Spreadsheet Documents - Working with Spreadsheets - Navigating*)
- Service `com.sun.star.sheet.SheetRangesQuery` (see *8.3.3 Spreadsheet Documents - Working with Spreadsheets - Navigating*)
- Interface `com.sun.star.util.XIndent` (see *8.3.2 Spreadsheet Documents - Working with Spreadsheets - Formatting*)
- Interface `com.sun.star.sheet.XSheetOperation` (see *8.3.1 Spreadsheet Documents - Working with Spreadsheets - Document Structure - Cell Ranges*)
- Interface `com.sun.star.chart.XChartDataArray` (see *10 Charts*)

The interfaces `com.sun.star.sheet.XSheetCellRangeContainer` and `com.sun.star.sheet.XSheetCellRanges` support basic handling of cell range collections.

- The method `getRangeAddressesAsString()` returns the string representation of all cell ranges.
- The method `getRangeAddresses()` returns a sequence with all cell range addresses.

The interface `com.sun.star.sheet.XSheetCellRangeContainer` is derived from the interface `com.sun.star.sheet.XSheetCellRanges` to insert and remove cell ranges.

- The methods `addRangeAddress()` and `addRangeAddresses()` insert one or more ranges into the collection. If the `boolean` parameter bMergeRanges is set to `true`, the methods try to merge the new range(s) with the ranges of the collection.
- The methods `removeRangeAddress()` and `removeRangeAddresses()` remove existing ranges from the collection. Only ranges that are contained in the collection are removed. The methods do not try to shorten a range.

The interface `com.sun.star.sheet.XSheetCellRanges` implements methods for access to cells and cell ranges:

- The method `getCells()` returns the interface `com.sun.star.container.XEnumerationAccess` of a cell collection. The service `com.sun.star.sheet.Cells` is discussed below. This collection contains the cell addresses of non-empty cells in all cell ranges.

The service `com.sun.star.sheet.Cells` represents a collection of cells.

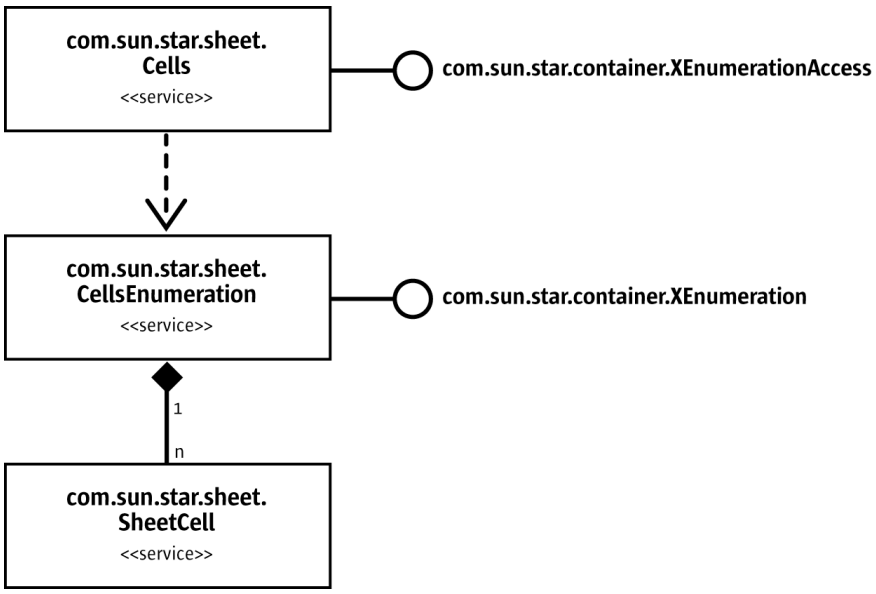


Illustration 92: Cell collections

The following example demonstrates the usage of cell range collections and cell collections. (Spreadsheet/SpreadsheetSample.java)

```
/** All samples regarding cell range collections. */
public void doCellRangesSamples(com.sun.star.sheet.XSpreadsheetDocument xDocument)
        throws RuntimeException, Exception {

    // Create a new cell range container
    com.sun.star.lang.XMultiServiceFactory xDocFactory =
        (com.sun.star.lang.XMultiServiceFactory) UnoRuntime.queryInterface(
            com.sun.star.lang.XMultiServiceFactory.class, xDocument);
    com.sun.star.sheet.XSheetCellRangeContainer xRangeCont =
        (com.sun.star.sheet.XSheetCellRangeContainer) UnoRuntime.queryInterface(
            com.sun.star.sheet.XSheetCellRangeContainer.class,
            xDocFactory.createInstance("com.sun.star.sheet.SheetCellRanges"));

    // Insert ranges
    insertRange(xRangeCont, 0, 0, 0, 0, 0, false);    // A1:A1
    insertRange(xRangeCont, 0, 0, 1, 0, 2, true);     // A2:A3
    insertRange(xRangeCont, 0, 1, 0, 1, 2, false);    // B1:B3

    // Query the list of filled cells
    System.out.print("All filled cells: ");
    com.sun.star.container.XEnumerationAccess xCellsEA = xRangeCont.getCells();
    com.sun.star.container.XEnumeration xEnum = xCellsEA.createEnumeration();
    while (xEnum.hasMoreElements()) {
        Object aCellObj = xEnum.nextElement();
        com.sun.star.sheet.XCellAddressable xAddr = (com.sun.star.sheet.XCellAddressable)
            UnoRuntime.queryInterface(com.sun.star.sheet.XCellAddressable.class, aCellObj);
        com.sun.star.table.CellAddress aAddr = xAddr.getCellAddress();
        System.out.print(getCellAddressString(aAddr.Column, aAddr.Row) + " ");
    }
    System.out.println();
}


/** Inserts a cell range address into a cell range container and prints a message.
    @param xContainer  The com.sun.star.sheet.XSheetCellRangeContainer interface of the container.
    @param nSheet  Index of sheet of the range.
    @param nStartCol  Index of first column of the range.
    @param nStartRow  Index of first row of the range.
    @param nEndCol  Index of last column of the range.
    @param nEndRow  Index of last row of the range.
    @param bMerge  Determines whether the new range should be merged with the existing ranges.
 */
private void insertRange(
        com.sun.star.sheet.XSheetCellRangeContainer xContainer,
        int nSheet, int nStartCol, int nStartRow, int nEndCol, int nEndRow,
        boolean bMerge) throws RuntimeException, Exception {
    com.sun.star.table.CellRangeAddress aAddress = new com.sun.star.table.CellRangeAddress();
    aAddress.Sheet = (short)nSheet;
    aAddress.StartColumn = nStartCol;
    aAddress.StartRow = nStartRow;
```

```
    aAddress.EndColumn = nEndCol;
    aAddress.EndRow = nEndRow;
    xContainer.addRangeAddress(aAddress, bMerge);
    System.out.println(
        "Inserting " + (bMerge ? "   with" : "without") + " merge,"
        + " result list: " + xContainer.getRangeAddressesAsString());
}
```

Columns and Rows

Collection of table columns:

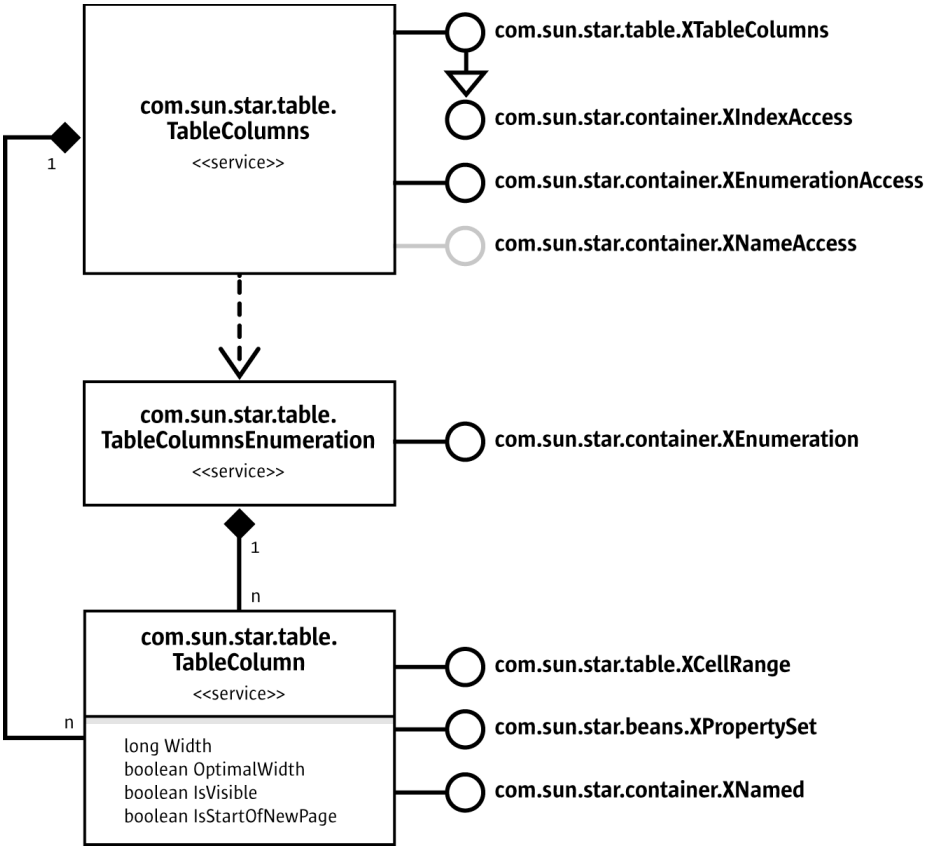


Illustration 93: Collection of table columns

Collection of table rows:

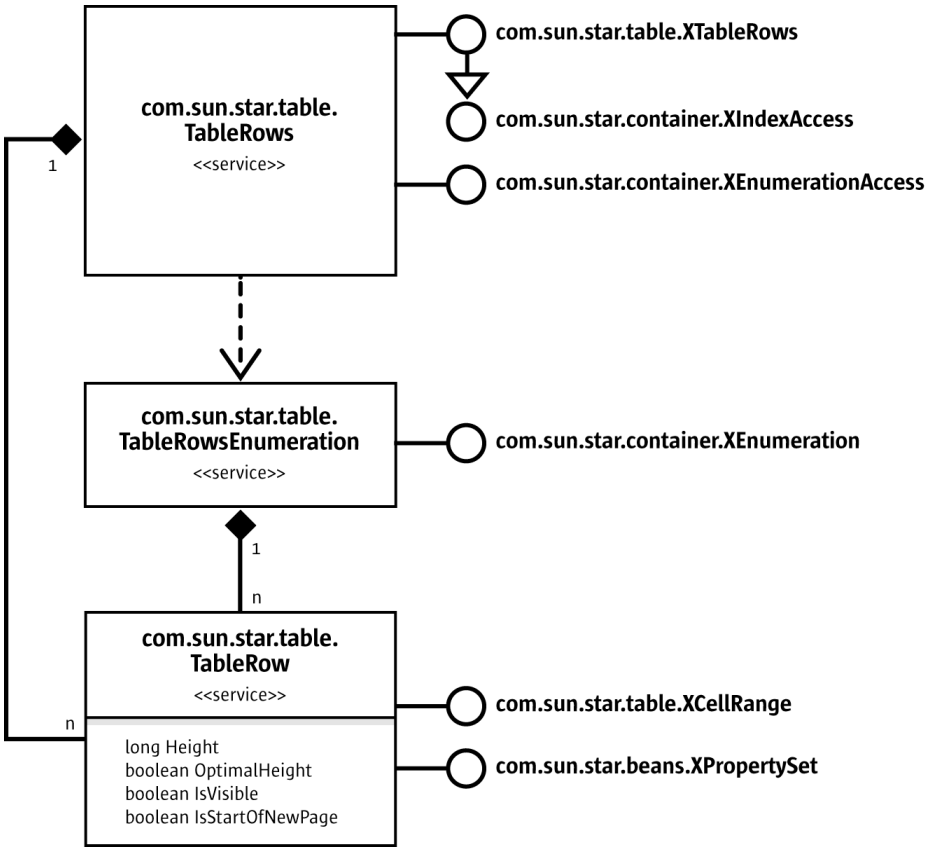


Illustration 94: Collection of table rows

The services `com.sun.star.table.TableColumns` and `com.sun.star.table.TableRows` represent collections of all columns and rows of a table. It is possible to access cells of columns and rows, and insert and remove columns and rows using the interfaces `com.sun.star.table.XTableColumns` and `com.sun.star.table.XTableRows` that are derived from `com.sun.star.container.XIndexAccess`. The method `createEnumeration()` of the interface `com.sun.star.container.XEnumerationAccess` creates an enumeration of all columns or rows. The interface `com.sun.star.container.XNameAccess` accesses columns through their names. The implementation of this interface is optional.

A single column or row is represented by the services `com.sun.star.table.TableColumn` and `com.sun.star.table.TableRow`. They implement the interfaces `com.sun.star.table.XCellRange` that provide access to the cells and `com.sun.star.beans.XPropertySet` for modifying settings. Additionally, the service TableColumn implements the interface `com.sun.star.container.XNamed`. It provides the method `getName()` that returns the name of a column. Changing the name of a column is not supported.



The interface `com.sun.star.container.XIndexAccess` returns columns and rows relative to the cell range (index 0 is always the first column or row of the cell range). But the interface `com.sun.star.container.XNameAccess` returns columns with their real names, regardless of the cell range.

In the following example, `xColumns` is an interface of a collection of columns, `xRows` is an interface of a collection of rows, and `xRange` is the range formed by the columns and rows. (Spreadsheet/GeneralTableSample.java)

```
com.sun.star.beans.XPropertySet xPropSet = null;

// *** Modifying COLUMNS and ROWS ***
// Get column C by index (interface XIndexAccess).
Object aColumnObj = xColumns.getByIndex(2);
xPropSet = (com.sun.star.beans.XPropertySet)
    UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, aColumnObj);
xPropSet.setPropertyValue("Width", new Integer(5000));

// Get the name of the column.
com.sun.star.container.XNamed xNamed = (com.sun.star.container.XNamed)
    UnoRuntime.queryInterface(com.sun.star.container.XNamed.class, aColumnObj);
aText = "The name of this column is " + xNamed.getName() + ".";
xRange.getCellByPosition(2, 2).setFormula(aText);

// Get column D by name (interface XNameAccess).
com.sun.star.container.XNameAccess xColumnsName = (com.sun.star.container.XNameAccess)
    UnoRuntime.queryInterface(com.sun.star.container.XNameAccess.class, xColumns);

aColumnObj = xColumnsName.getByName("D");
xPropSet = (com.sun.star.beans.XPropertySet)
    UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, aColumnObj);
xPropSet.setPropertyValue("IsVisible", new Boolean(false));

// Get row 7 by index (interface XIndexAccess)
Object aRowObj = xRows.getByIndex(6);
xPropSet = (com.sun.star.beans.XPropertySet)
    UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, aRowObj);
xPropSet.setPropertyValue("Height", new Integer(5000));

// Create a cell series with the values 1 ... 7.
for (int nRow = 8; nRow < 15; ++nRow)
    xRange.getCellByPosition( 0, nRow ).setValue( nRow - 7 );
// Insert a row between 1 and 2
xRows.insertByIndex(9, 1);
// Delete the rows with the values 3 and 4.
xRows.removeByIndex(11, 2);
```

8.3.2 Formatting

Cell Formatting

In cells, cell ranges, table rows, table columns and cell ranges collections, the cells are formatted through the service `com.sun.star.table.CellProperties`. These properties are accessible through the interface `com.sun.star.beans.XPropertySet` that is supported by all the objects mentioned above. The service contains all properties that describe the cell formatting of the cell range, such as the cell background color, borders, the number format and the cell alignment. Changing the property values affects all cells of the object being formatted.

The cell border style is stored in the struct `com.sun.star.table.TableBorder`. A cell range contains six different kinds of border lines: upper, lower, left, right, horizontal inner, and vertical inner line. Each line is represented by a struct `com.sun.star.table.BorderLine` that contains the line style and color. The `boolean` members `Is...LineValid` specifies the validity of the `...Line` members containing the line style. If the property contains the value `true`, the line style is equal in all cells that include the line. The style is contained in the `...Line` struct. The value `false` means the cells are formatted differently and the content of the `...Line` struct is undefined. When changing the border property, these boolean values determine if the lines are changed to the style contained in the respective `...Line` struct.

Character and Paragraph Format

The following services of a cell range contain properties for the character style and paragraph format:

- Service `com.sun.star.style.ParagraphProperties`

- Service `com.sun.star.style.CharacterProperties`
- Service `com.sun.star.style.CharacterPropertiesAsian`
- Service `com.sun.star.style.CharacterPropertiesComplex`

The chapter *7.3.2 Text Documents - Working with Text Documents - Formatting* contains a description of these properties.

This example formats a given cell range `xCellRange`: (Spreadsheet/SpreadsheetSample.java)

```
// --- Change cell range properties. ---
com.sun.star.beans.XPropertySet xPropSet = (com.sun.star.beans.XPropertySet)
    UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, xCellRange);

// from com.sun.star.styles.CharacterProperties
xPropSet.setPropertyValue("CharColor", new Integer(0x003399));
xPropSet.setPropertyValue("CharHeight", new Float(20.0));

// from com.sun.star.styles.ParagraphProperties
xPropSet.setPropertyValue("ParaLeftMargin", new Integer(500));

// from com.sun.star.table.CellProperties
xPropSet.setPropertyValue("IsCellBackgroundTransparent", new Boolean(false));
xPropSet.setPropertyValue("CellBackColor", new Integer(0x99CCFF));
```

The code below changes the character and paragraph formatting of a cell. Assume that `xCell` is a `com.sun.star.table.XCell` interface of a spreadsheet cell.
(Spreadsheet/SpreadsheetSample.java)

```
// --- Change cell properties. ---
com.sun.star.beans.XPropertySet xPropSet = (com.sun.star.beans.XPropertySet)
    UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, xCell);

// from styles.CharacterProperties
xPropSet.setPropertyValue("CharColor", new Integer(0x003399));
xPropSet.setPropertyValue("CharHeight", new Float(20.0));

// from styles.ParagraphProperties
xPropSet.setPropertyValue("ParaLeftMargin", new Integer(500));

// from table.CellProperties
xPropSet.setPropertyValue("IsCellBackgroundTransparent", new Boolean(false));
xPropSet.setPropertyValue("CellBackColor", new Integer(0x99CCFF));
```

Indentation

The methods of the interface `com.sun.star.util.XIndent` change the left indentation of the cell contents. This interface is supported by cells, cell ranges and collections of cell ranges. The indentation is incremental and decremental, independent for each cell.

- The method `decrementIndent()` reduces the indentation of each cell by 1.
- The method `incrementIndent()` enlarges the indentation of each cell by 1.

The following sample shows how to increase the cell indentation by 1.
(Spreadsheet/SpreadsheetSample.java)

```
// --- Change indentation. ---
com.sun.star.util.XIndent xIndent = (com.sun.star.util.XIndent)
    UnoRuntime.queryInterface(com.sun.star.util.XIndent.class, xCellRange);
xIndent.incrementIndent();
```



Due to a bug, this interface does not work in the current implementation. Workaround: Use the paragraph property `ParaIndent`.

Equally Formatted Cell Ranges

It is possible to get collections of all equally formatted cell ranges contained in a source cell range.

Cell Format Ranges

The service `com.sun.star.sheet.CellFormatRanges` represents a collection of equally formatted cell ranges. The cells inside of a cell range of the collection have the same formatting attributes. All cells of the source range are contained in one of the ranges. If there is a non-rectangular, equal-formatted range, it is split into several rectangular ranges.

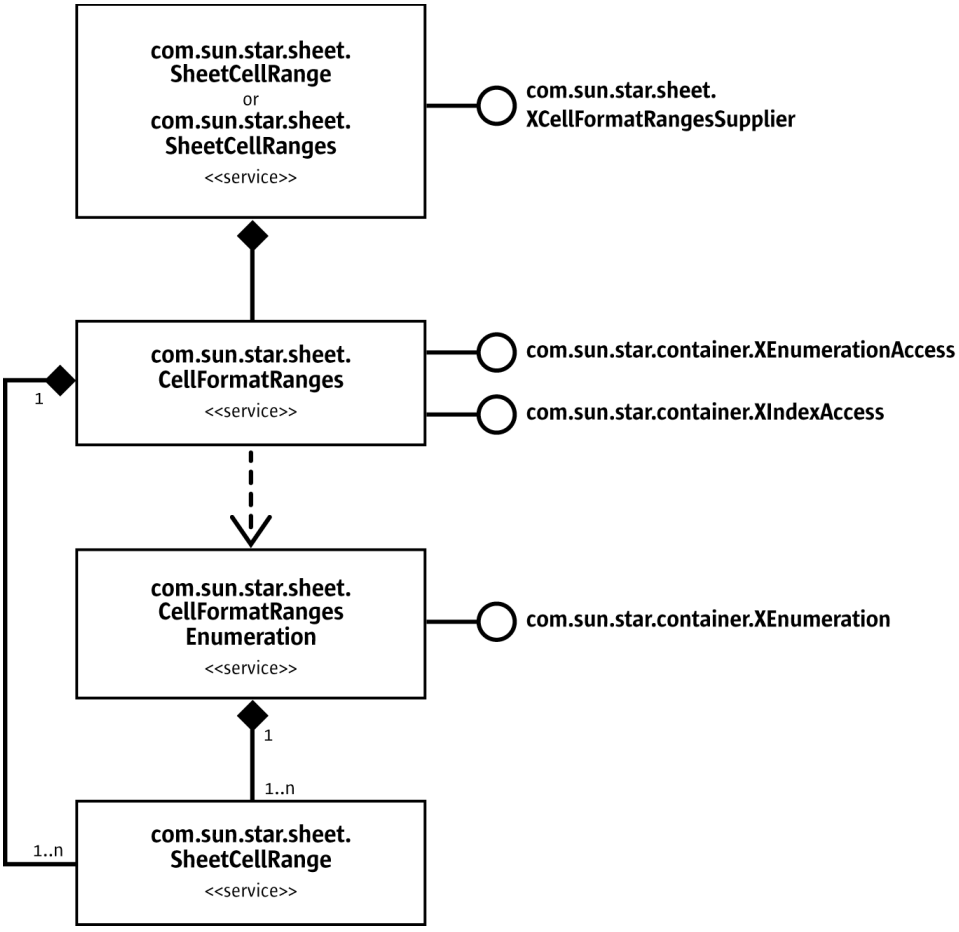


Illustration 95: Cell Format Ranges

Unique Cell Format Ranges

The service `com.sun.star.sheet.UniqueCellFormatRanges` represents, similar to Cell Format Ranges above, a collection of equally formatted cell ranges, but this collection contains cell range container objects (service `com.sun.star.sheet.SheetCellRanges`) that contain the cell ranges. The cells of all ranges inside of a cell range container are equally formatted. The formatting attributes of a range container differ from each other range container. All equally formatted ranges are consolidated into one container.

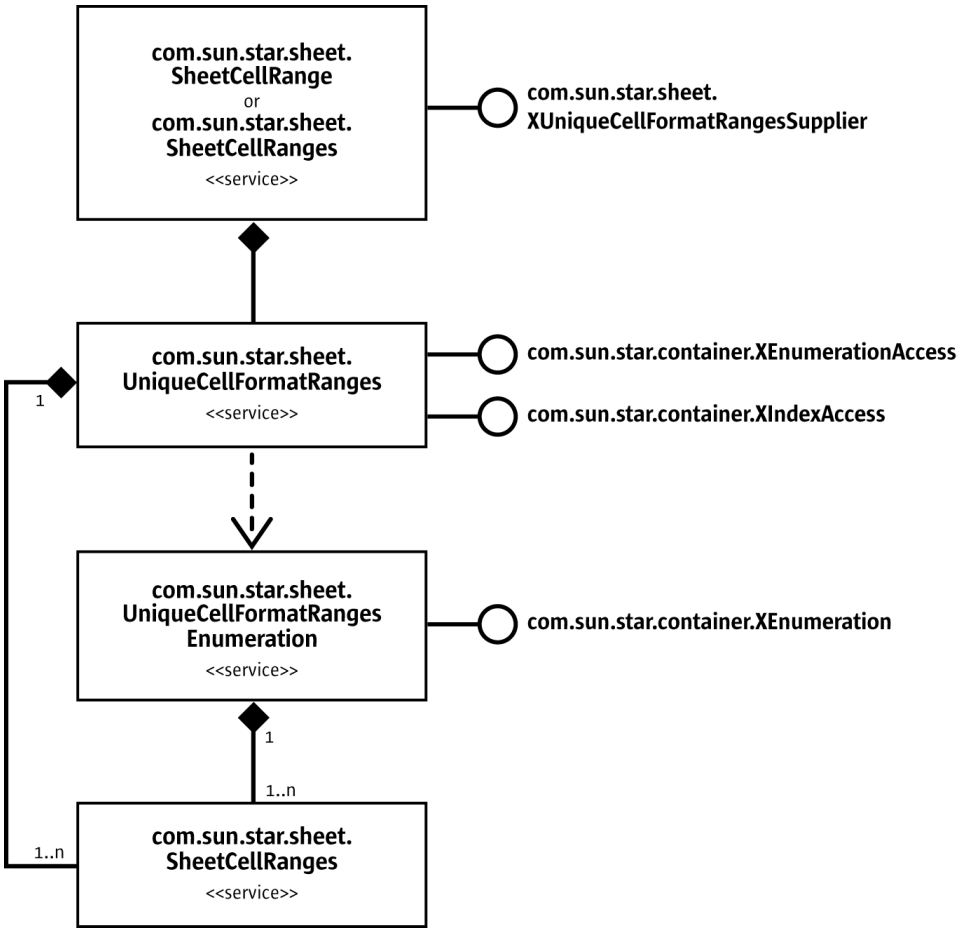


Illustration 96: UniqueCellFormatRanges

In the following example, the cells have two different background colors. The formatted ranges of the range A1:G3 are queried in both described ways.

	A	B	C	D	E	F	G
1							
2							
3							

A `com.sun.star.sheet.CellFormatRanges` object contains the following ranges: A1:C2, D1:G1, D2:F2, G2:G2, and A3:G3.

A `com.sun.star.sheet.UniqueCellFormatRanges` object contains two `com.sun.star.sheet.SheetCellRanges` range collections. One collection contains the white ranges, that is, A1:C2, D1:G1, G2:G2, and the other collection, the gray ranges, that is, D2:F2, A3:G3.

The following code is an example of accessing the formatted ranges in Java. The `getCellRangeAddressString` is a helper method that returns the range address as a string. (Spreadsheet/SpreadsheetSample.java)

```
/** All samples regarding formatted cell ranges. */
public void doFormattedCellRangesSamples(com.sun.star.sheet.XSpreadsheet xSheet)
        throws RuntimeException, Exception {
    // All ranges in one container
    xCellRange = xSheet.getCellRangeByName("A1:G3");
    System.out.println("Service CellFormatRanges:");
    com.sun.star.sheet.XCellFormatRangesSupplier xFormatSupp =
        (com.sun.star.sheet.XCellFormatRangesSupplier) UnoRuntime.queryInterface(
            com.sun.star.sheet.XCellFormatRangesSupplier.class, xCellRange);
    com.sun.star.container.XIndexAccess xRangeIA = xFormatSupp.getCellFormatRanges();
    System.out.println( getCellRangeListString(xRangeIA));

    // Ranges sorted in SheetCellRanges containers
    System.out.println("\nService UniqueCellFormatRanges:");
    com.sun.star.sheet.XUniqueCellFormatRangesSupplier xUniqueFormatSupp =
        (com.sun.star.sheet.XUniqueCellFormatRangesSupplier) UnoRuntime.queryInterface(
            com.sun.star.sheet.XUniqueCellFormatRangesSupplier.class, xCellRange);
    com.sun.star.container.XIndexAccess xRangesIA = xUniqueFormatSupp.getUniqueCellFormatRanges();
    int nCount = xRangesIA.getCount();
    for (int nIndex = 0; nIndex < nCount; ++nIndex) {
        Object aRangesObj = xRangesIA.getByIndex(nIndex);
        xRangeIA = (com.sun.star.container.XIndexAccess) UnoRuntime.queryInterface(
            com.sun.star.container.XIndexAccess.class, aRangesObj);
        System.out.println(
            "Container " + (nIndex + 1) + ": " + getCellRangeListString(xRangeIA));
    }
}


/** Returns a list of addresses of all cell ranges contained in the collection.
    @param xRangesIA  The XIndexAccess interface of the collection.
    @return  A string containing the cell range address list.
 */
private String getCellRangeListString( com.sun.star.container.XIndexAccess xRangesIA )
        throws RuntimeException, Exception {
    String aStr = "";
    int nCount = xRangesIA.getCount();
    for (int nIndex = 0; nIndex < nCount; ++nIndex) {
        if (nIndex > 0)
            aStr += " ";
        Object aRangeObj = xRangesIA.getByIndex(nIndex);
        com.sun.star.sheet.XSheetCellRange xCellRange = (com.sun.star.sheet.XSheetCellRange)
            UnoRuntime.queryInterface(com.sun.star.sheet.XSheetCellRange.class, aRangeObj);
        aStr += getCellRangeAddressString(xCellRange, false);
    }
    return aStr;
}
```

Table Auto Formats

Table auto formats are used to apply different formats to a cell range. A table auto format is a collection of cell styles used to format all cells of a range. The style applied is dependent on the position of the cell.

The table auto format contains separate information about four different row types and four different column types:

- First row (header), first data area row, second data area row, last row (footer)
- First column, first data area column, second data area column, last column

The row or column types for the data area (between first and last row/column) are repeated in sequence. Each cell of the formatted range belongs to one of the row types and column types, resulting in 16 different auto-format fields. In the example below, the highlighted cells have the formatting of the first data area row and last column field. Additionally, this example shows the indexes of all the auto format fields. These indexes are used to access the field with the interface `com.sun.star.container.XIndexAccess`.

	First column	First data area column	Second data area column	First data area column	Last Column
First row (header)	0	1	2	1	3
First data area row	4	5	6	5	**7**
Second data area row	8	9	10	9	11
First data area row	4	5	6	5	**7**
Second data area row	8	9	10	9	11
Last row (footer)	12	13	14	13	15

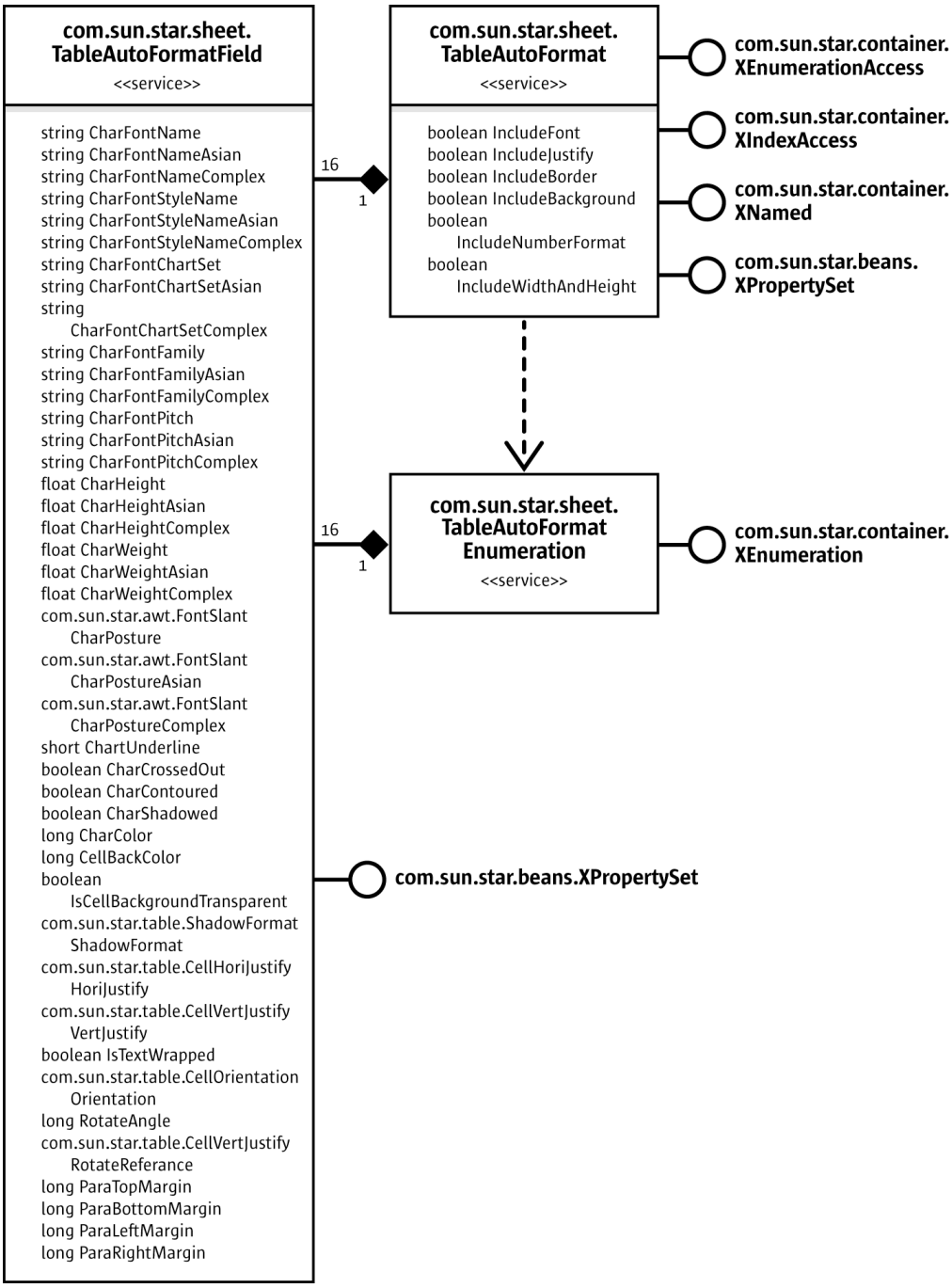


Illustration 97: TableAutoFormat

A table auto format is represented by the service `com.sun.star.sheet.TableAutoFormat`. It contains exactly 16 auto format fields (service `com.sun.star.sheet.TableAutoFormatField`). Each auto format field contains all properties of a single cell.

The cell range interface `com.sun.star.table.XAutoFormattable` contains the method `autoFormat()` that applies a *table auto format* to a cell range. The cell range must have a size of at least 3x3 cells. The `boolean` properties of the table auto format determine the formatting properties are copied to the cells. The default setting of all the properties is `true`.



In the current implementation it is not possible to modify the cell borders of a table auto format (the property `TableBorder` is missing). Nevertheless, the property `IncludeBorder` controls whether the borders of default auto formats are applied to the cells.

The collection of all table auto formats is represented by the service `com.sun.star.sheet.TableAutoFormats`. There is only one instance of this collection in the whole application. It contains all default and user-defined auto formats that are used in spreadsheets and tables of the word-processing application. It is possible to iterate through all table auto formats with an enumeration, or to access them directly using their index or their name.

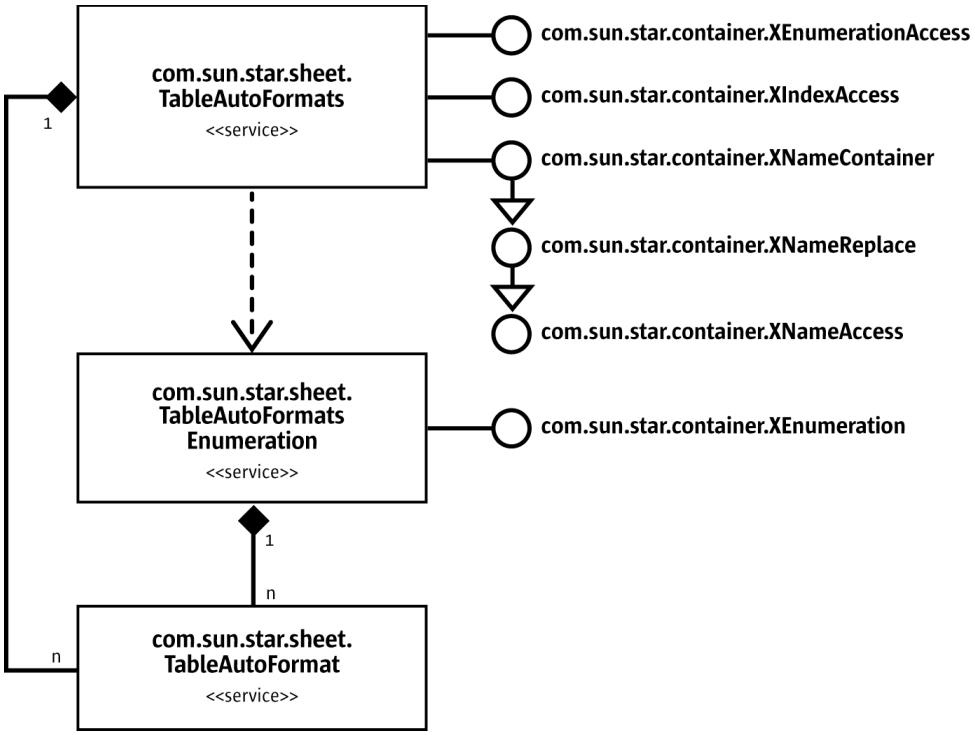


Illustration 98: TableAutoFormats

The following example shows how to insert a new table auto format, fill it with properties, apply it to a cell range and remove it from the format collection. (Spreadsheet/SpreadsheetSample.java)

```
public void doAutoFormatSample(
        com.sun.star.lang.XMultiServiceFactory xServiceManager,
        com.sun.star.sheet.XSpreadsheetDocument xDocument) throws RuntimeException, Exception {
    // get the global collection of table auto formats, use global service manager
    Object aAutoFormatsObj = xServiceManager.createInstance("com.sun.star.sheet.TableAutoFormats");
    com.sun.star.container.XNameContainer xAutoFormatsNA = (com.sun.star.container.XNameContainer)
        UnoRuntime.queryInterface(com.sun.star.container.XNameContainer.class, aAutoFormatsObj);

    // create a new table auto format and insert into the container
    String aAutoFormatName =  "Temp_Example";
    boolean bExistsAlready = xAutoFormatsNA.hasByName(aAutoFormatName);
    Object aAutoFormatObj = null;
    if (bExistsAlready)
        // auto format already exists -> use it
        aAutoFormatObj = xAutoFormatsNA.getByName(aAutoFormatName);
    else {
        // create a new auto format (with document service manager!)
        com.sun.star.lang.XMultiServiceFactory xDocServiceManager =
            (com.sun.star.lang.XMultiServiceFactory) UnoRuntime.queryInterface(
                com.sun.star.lang.XMultiServiceFactory.class, xDocument);
        aAutoFormatObj = xDocServiceManager.createInstance("com.sun.star.sheet.TableAutoFormat");
        xAutoFormatsNA.insertByName(aAutoFormatName, aAutoFormatObj);
    }
    // index access to the auto format fields
    com.sun.star.container.XIndexAccess xAutoFormatIA = (com.sun.star.container.XIndexAccess)
        UnoRuntime.queryInterface(com.sun.star.container.XIndexAccess.class, aAutoFormatObj);

    // set properties of all auto format fields
    for (int nRow = 0; nRow < 4; ++nRow) {
        int nRowColor = 0;
        switch (nRow) {
            case 0:    nRowColor = 0x999999;   break;
            case 1:    nRowColor = 0xFFFFCC;   break;
            case 2:    nRowColor = 0xEEEEEE;   break;
            case 3:    nRowColor = 0x999999;   break;
        }

        for (int nColumn = 0; nColumn < 4; ++nColumn) {
            int nColor = nRowColor;
            if ((nColumn == 0) || (nColumn == 3))
                nColor -= 0x333300;

            // get the auto format field and apply properties
            Object aFieldObj = xAutoFormatIA.getByIndex(4 * nRow + nColumn);
            com.sun.star.beans.XPropertySet xPropSet = (com.sun.star.beans.XPropertySet)
                UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, aFieldObj);
            xPropSet.setPropertyValue("CellBackColor", new Integer(nColor));
        }
    }

    // set the auto format to the second spreadsheet
    com.sun.star.sheet.XSpreadsheets xSheets = xDocument.getSheets();
    com.sun.star.container.XIndexAccess xSheetsIA = (com.sun.star.container.XIndexAccess)
        UnoRuntime.queryInterface(com.sun.star.container.XIndexAccess.class, xSheets);

    com.sun.star.sheet.XSpreadsheet xSheet =
        (com.sun.star.sheet.XSpreadsheet) xSheetsIA.getByIndex(1);

    com.sun.star.table.XCellRange xCellRange = xSheet.getCellRangeByName("A5:H25");
    com.sun.star.table.XAutoFormattable xAutoForm = (com.sun.star.table.XAutoFormattable)
        UnoRuntime.queryInterface(com.sun.star.table.XAutoFormattable.class, xCellRange);

    xAutoForm.autoFormat(aAutoFormatName);

    // remove the auto format
    if (!bExistsAlready)
        xAutoFormatsNA.removeByName(aAutoFormatName);
}
```

Conditional Formats

A cell can be formatted automatically with a conditional format, depending on its contents or the result of a formula. A conditional format consists of several condition entries that contain the condition and name of a cell style. The style of the first met condition, `true` or "not zero", is applied to the cell.



Illustration 99: TableConditionalFormats

A cell or cell range object contains the properties `ConditionalFormat` and `ConditionalFormatLocal`. These properties return the interface `com.sun.star.sheet.XSheetConditionalEntries` of the conditional format container `com.sun.star.sheet.TableConditionalFormat`. The objects of both properties are equal, except for the representation of formulas. The `ConditionalFormatLocal` property uses function names in the current language.



After a conditional format is changed, it has to be reinserted into the property set of the cell or cell range.

A condition entry of a conditional format is represented by the service `com.sun.star.sheet.TableConditionalEntry`. It implements two interfaces:

- The interface `com.sun.star.sheet.XSheetCondition` gets and sets the operator, the first and second formula and the base address for relative references.
- The interface `com.sun.star.sheet.XSheetConditionalEntry` gets and sets the cell style name.

The service `com.sun.star.sheet.TableConditionalFormat` contains all format conditions and returns `com.sun.star.sheet.TableConditionalEntry` objects. The interface `com.sun.star.sheet.XSheetConditionalEntries` inserts new conditions and removes them.

- The method `addNew()` inserts a new condition. It expects a sequence of `com.sun.star.beans.PropertyValue` objects. The following properties are supported:

- `Operator`: A `com.sun.star.sheet.ConditionOperator` constant describing the operation to perform.
- `Formula1` and `Formula2`: Strings that contain the values or formulas to evaluate. `Formula2` is used only if the property `Operator` contains `BETWEEN` or `NOT_BETWEEN`.
- `SourcePosition`: A `com.sun.star.table.CellAddress` struct that contains the base address for relative cell references in formulas.
- `StyleName`: The name of the cell style to apply.

- The methods `removeByIndex()` removes the condition entry at the specified position.
- The method `clear()` removes all condition entries.

The following example applies a conditional format to a cell range. It uses the cell style "MyNewCellStyle" that is applied to each cell containing a value greater than 1. The `xSheet` is the `com.sun.star.sheet.XSpreadsheet` interface of a spreadsheet.
(Spreadsheet/SpreadsheetSample.java)

```
// get the conditional format object of the cell range
com.sun.star.table.XCellRange xCellRange = xSheet.getCellRangeByName("A1:B10");
com.sun.star.beans.XPropertySet xPropSet = (com.sun.star.beans.XPropertySet)
    UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, xCellRange);
com.sun.star.sheet.XSheetConditionalEntries xEntries =
    (com.sun.star.sheet.XSheetConditionalEntries) xPropSet.getPropertyValue("ConditionalFormat");

// create a condition and apply it to the range
com.sun.star.beans.PropertyValue[] aCondition = new com.sun.star.beans.PropertyValue[3];
aCondition[0] = new com.sun.star.beans.PropertyValue();
aCondition[0].Name  = "Operator";
aCondition[0].Value = com.sun.star.sheet.ConditionOperator.GREATER;
aCondition[1] = new com.sun.star.beans.PropertyValue();
aCondition[1].Name  = "Formula1";
aCondition[1].Value = "1";
aCondition[2] = new com.sun.star.beans.PropertyValue();
aCondition[2].Name  = "StyleName";
aCondition[2].Value = "MyNewCellStyle";
xEntries.addNew(aCondition);
xPropSet.setPropertyValue("ConditionalFormat", xEntries);
```

8.3.3 Navigating

Unlike other document models that provide access to their content by content suppliers, the spreadsheet document contains properties that allow direct access to various containers.



This design inconsistency may be changed in future versions. The properties remain for compatibility.

The properties allow access to various containers:

- `NamedRanges`: The container with all the named ranges. See *8.3.3 Spreadsheet Documents - Working with Spreadsheets - Navigating - Named Ranges*.
- `ColumnLabelRanges` and `RowLabelRanges`: Containers with row labels and column labels. See *8.3.3 Spreadsheet Documents - Working with Spreadsheets - Navigating - Label Ranges*.
- `DatabaseRanges`: The container with all database ranges. See *8.3.5 Spreadsheet Documents - Working with Spreadsheets - Database Operations - Database Ranges*.
- `SheetLinks`, `AreaLinks` and `DDELinks`: Containers with external links. See *8.3.6 Spreadsheet Documents - Working with Spreadsheets - Linking External Data - Sheet Links*.

Cell Cursor

A *cell cursor* is a cell range with extended functionality and is represented by the service `com.sun.star.sheet.SheetCellCursor`. With a cell cursor it is possible to move through a cell range. Each table can contain only one cell cursor.

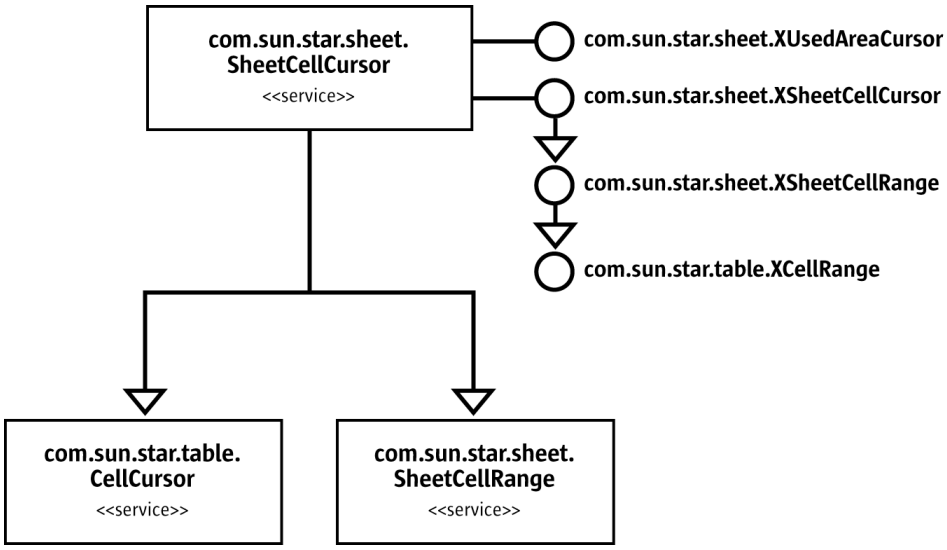


Illustration 100: Cell cursor

It implements all interfaces described in *8.3.1 Spreadsheet Documents - Working with Spreadsheets - Document Structure - Cell Ranges* and the basic cursor interfaces of the service `com.sun.star.table.CellCursor` that represents the cell or cell range cursor of a table.

The interface `com.sun.star.sheet.XSpreadsheet` of a spreadsheet creates the cell cursors. The methods return the interface `com.sun.star.sheet.XSheetCellCursor` of the cursor. It is derived from the interface `com.sun.star.sheet.XSheetCellRange` that provides access to cells and cell ranges. Refer to *8.3.1 Spreadsheet Documents - Working with Spreadsheets - Document Structure - Cell Ranges* for additional information.

- The method `createCursor()` creates a cursor that spans over the whole spreadsheet.
- The method `createCursorByRange()` creates a cursor that spans over the given cell range.

The `SheetCellCursor` includes the `CellCursor` service from the table module:

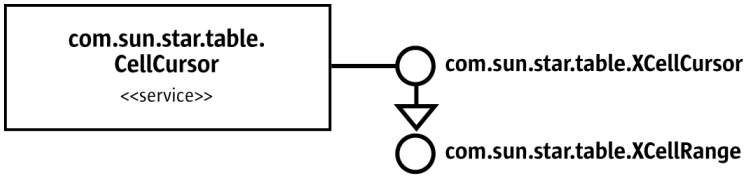


Illustration 101: Table cell cursor

Cursor Movement

The service `com.sun.star.table.CellCursor` implements the interface `com.sun.star.table.XCellCursor` that provides methods to move to specific cells of a cell range. This interface is derived from `com.sun.star.table.XCellRange` so all methods that access single cells can be used.

Methods of `com.sun.star.table.XCellCursor`	
`gotoStart()`	Moves to the first filled cell. This cell may be outside of the current range of the cursor.
`gotoEnd()`	Moves to the last filled cell. This cell may be outside of the current range of the cursor.
`gotoOffset()`	Moves the cursor relative to the current position, even if the cursor is a range.
`gotoPrev()`	Moves the cursor to the latest available unprotected cell. In most cases,this is the cell to the left of the current cell.
`gotoNext()`	Moves the cursor to the next available unprotected cell. In most cases,this is the cell to the right of the current cell.

The following example shows how to modify a cell beyond a filled area.The `xCursor` may be an initialized cell cursor. (Spreadsheet/GeneralTableSample.java)

```
// *** Use the cell cursor to add some data below of the filled area ***
// Move to the last filled cell.
xCursor.gotoEnd();
// Move one row down.
xCursor.gotoOffset(0, 1);
xCursor.getCellByPosition(0, 0).setFormula("Beyond of the last filled cell.");
```

The interface `com.sun.star.sheet.XSheetCellCursor` sets the cursor to specific ranges in the sheet.

- The method `collapseToCurrentRegion()` expands the cursor to the shortest cell range filled with any data. A few examples from the spreadsheet below are: the cursor C2:C2 expands to B2:D3, cursor C1:C2 expands to B1:D3 and cursor A1:D4 is unchanged.

	A	B	C	D	E	F	G
1							
2			1	3		{=C2:D3}	{=C2:D3}
3		Text	2	4		{=C2:D3}	{=C2:D3}
4							

- The method `collapseToCurrentArray()` expands or shortens the cursor range to an array formula range. This works only if the top-left cell of the current cursor contains an array formula. An example using the spreadsheet above: All the cursors with a top-left cell located in the range F2:G3 are modified to this array formula range, F2:F2 or G2:G4.
- The method `collapseToMergedArea()` expands the current cursor range so that all merged cell ranges intersecting the current range fit completely.
- The methods `expandToEntireColumns()` and `expandToEntireRows()` expand the cursor range so that it contains all cells of the columns or rows of the current range.
- The method `collapseToSize()` resizes the cursor range to the given dimensions. The start address of the range is left unmodified. To move the cursor range without changing the current size, use the method `gotoOffset()` from the interface `com.sun.star.table.XCellCursor`.



Some of the methods above have misleading names: `collapseToCurrentRegion()` and `collapse-ToMergedArea()` expand the cursor range,but never shorten it and `collapseToCurrentArray()` may expand or shorten the cursor range.

The following example tries to find the range of the array formula in cell F22.The `xSheet` is a `com.sun.star.sheet.XSpreadsheet` interface of a spreadsheet and `getCellRangeAddressString()` is a helper method that returns the range address as a string.
(Spreadsheet/SpreadsheetSample.java)

```
// --- find the array formula using a cell cursor ---
com.sun.star.table.XCellRange xRange = xSheet.getCellRangeByName("F22");
com.sun.star.sheet.XSheetCellRange xCellRange = (com.sun.star.sheet.XSheetCellRange)
    UnoRuntime.queryInterface(com.sun.star.sheet.XSheetCellRange.class, xRange);
com.sun.star.sheet.XSheetCellCursor xCursor = xSheet.createCursorByRange(xCellRange);

xCursor.collapseToCurrentArray();
com.sun.star.sheet.XArrayFormulaRange xArray = (com.sun.star.sheet.XArrayFormulaRange)
    UnoRuntime.queryInterface(com.sun.star.sheet.XArrayFormulaRange.class, xCursor);
System.out.println(
    "Array formula in " + getCellRangeAddressString(xCursor, false)
    + " contains formula " + xArray.getArrayFormula());
```

Used Area

The cursor interface `com.sun.star.sheet.XUsedAreaCursor` contains methods to locate the used area of the entire sheet. The used area is the smallest cell range that contains all cells of the spreadsheet with any contents, such as values, text, and formulas, or visible formatting, such as borders and background color. In the following example, `xSheet` is a `com.sun.star.sheet.XSpreadsheet` interface of a spreadsheet.
(Spreadsheet/SpreadsheetSample.java)

```
// --- Find the used area ---
com.sun.star.sheet.XSheetCellCursor xCursor = xSheet.createCursor();
com.sun.star.sheet.XUsedAreaCursor xUsedCursor = (com.sun.star.sheet.XUsedAreaCursor)
    UnoRuntime.queryInterface(com.sun.star.sheet.XUsedAreaCursor.class, xCursor);
xUsedCursor.gotoStartOfUsedArea(false);
xUsedCursor.gotoEndOfUsedArea(true);
System.out.println("The used area is: " + getCellRangeAddressString(xCursor, true));
```

Referencing Ranges by Name

Cell ranges can be assigned a name that they may be addressed by in formulas. This is done with *named ranges*. Another way to use names for cell references in formulas is the automatic label lookup which is controlled using *label ranges*.

Named Ranges

A named range is a named formula expression, where a cell range is just one possible content. Thus, the content of a named range is always set as a string.

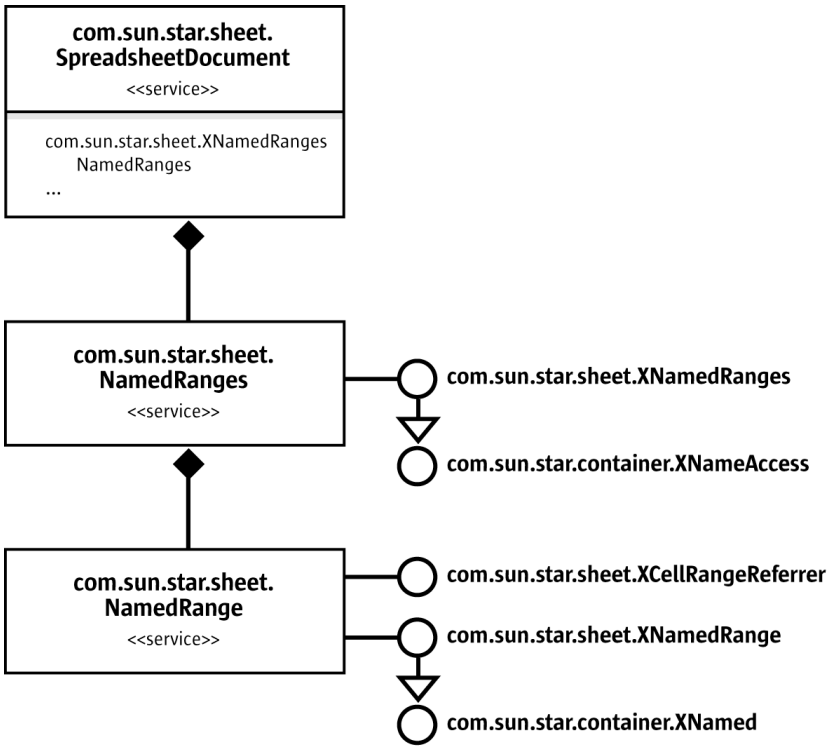


Illustration 102: Named ranges

The collection of named ranges is accessed using the document's `NamedRanges` property. A new named range is added by calling the `com.sun.star.sheet.XNamedRanges` interface's `addNewByName()` method. The method's parameters are:

- The name for the new named range.
- The content. This must be a string containing a valid formula expression. A commonly used type of expression is an absolute cell range reference like “$Sheet1.$A1:$C3”.

- A reference position for relative references. If the content contains relative cell references, and the named range is used in a formula, the references are adjusted for the formula's position. The reference position states which cell the references are relative to.
- The type of the named range that controls if the named range is included in some dialogs. The type must be a combination of the `com.sun.star.sheet.NamedRangeFlag` constants:
  - If the `FILTER_CRITERIA` bit is set, the named range is offered as a criteria range in the "Advanced Filter" dialog.
  - If the `PRINT_AREA`, `COLUMN_HEADER` or `ROW_HEADER` bit is set, the named range is selected as "Print range", "Columns to repeat" or "Rows to repeat" in the **Edit Print Ranges** dialog.

The `addNewFromTitles()` method creates named ranges from header columns or rows in a cell range. The `com.sun.star.sheet.Border` enum parameter selects which named ranges are created:

- If the value is `TOP`, a named range is created for each column of the cell range with the name taken from the range's first row, and the other cells of that column within the cell range as content.
- For `BOTTOM`, the names are taken from the range's last row.
- If the value is `LEFT`, a named range is created for each row of the cell range with the name taken from the range's first column, and the other cells of that row within the cell range as content.
- For `RIGHT`, the names are taken from the range's last column.

The `removeByName()` method is used to remove a named range. The `outputList()` method writes a list of all the named ranges into the document, starting at the specified cell position.

The `com.sun.star.sheet.NamedRange` service accesses an existing named range. The `com.sun.star.container.XNamed` interface changes the name, and the `com.sun.star.sheet.XNamedRange` interface changes the other settings. See the `addNewByName` description above for the meaning of the individual values.

If the content of the name is a single cell range reference, the `com.sun.star.sheet.XCellRangeReferrer` interface is used to access that cell range.

The following example creates a named range that calculates the sum of the two cells above the position where it is used. This is done by using the relative reference "G43:G44" with the reference position G45. Then, the example uses the named range in two formulas. (Spreadsheet/SpreadsheetSample.java)

```
// insert a named range
com.sun.star.beans.XPropertySet xDocProp = (com.sun.star.beans.XPropertySet)
    UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, xDocument);
Object aRangesObj = xDocProp.getPropertyValue("NamedRanges");
com.sun.star.sheet.XNamedRanges xNamedRanges = (com.sun.star.sheet.XNamedRanges)
    UnoRuntime.queryInterface(com.sun.star.sheet.XNamedRanges.class, aRangesObj);
com.sun.star.table.CellAddress aRefPos = new com.sun.star.table.CellAddress();
aRefPos.Sheet  = 0;
aRefPos.Column = 6;
aRefPos.Row    = 44;
xNamedRanges.addNewByName("ExampleName", "SUM(G43:G44)", aRefPos, 0);

// use the named range in formulas
xSheet.getCellByPosition(6, 44).setFormula("=ExampleName");
xSheet.getCellByPosition(7, 44).setFormula("=ExampleName");
```

Label Ranges

A label range consists of a label area containing the labels, and a data area containing the data that the labels address. There are label ranges for columns and rows of data, which are kept in two separate collections in the document.

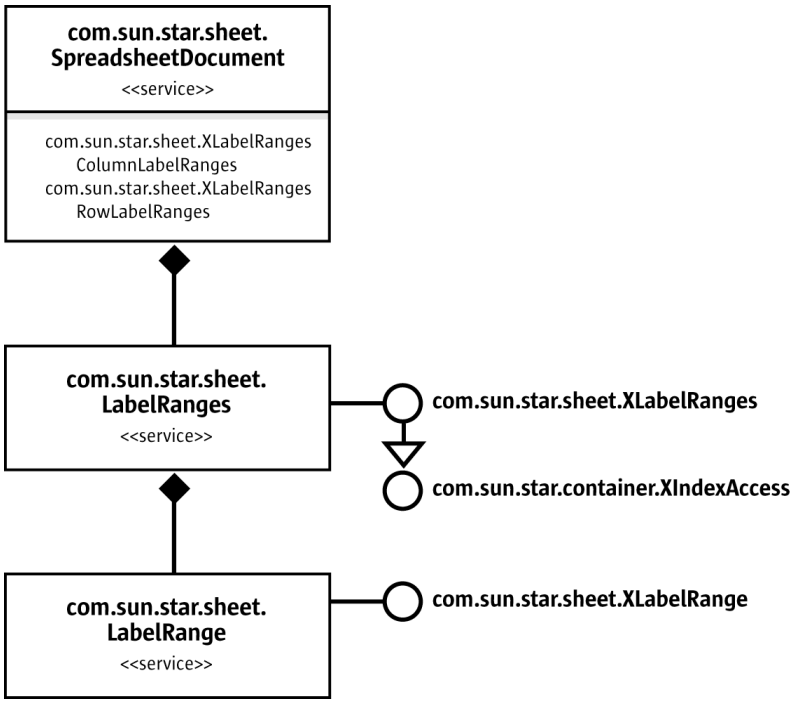


Illustration 103: Label Ranges

The `com.sun.star.sheet.LabelRanges` service contains the document's column label ranges or row label ranges, depending if the `ColumnLabelRanges` or `RowLabelRanges` property was used to get it. The `com.sun.star.sheet.XLabelRanges` interface's `addNew()` method is used to add a new label range, specifying the label area and data area. The `removeByIndex()` method removes a label range.

The `com.sun.star.sheet.LabelRange` service represents a single label range and contains the `com.sun.star.sheet.XLabelRange` interface to modify the label area and data area.

The following example inserts a column label range with the label area G48:H48 and the data area G49:H50, that is, the content of G48 is used as a label for G49:G50 and the content of H48 is used as a label for H49:H50, as shown in the two formulas the example inserts. (Spreadsheet/SpreadsheetSample.java)

```
com.sun.star.table.XCellRange xRange = xSheet.getCellRangeByPosition(6, 47, 7, 49);
com.sun.star.sheet.XCellRangeData xData = (com.sun.star.sheet.XCellRangeData)
    UnoRuntime.queryInterface(com.sun.star.sheet.XCellRangeData.class, xRange);
Object[][] aValues =
{
    {"Apples", "Oranges"},
    {new Double(5), new Double(7)},
    {new Double(6), new Double(8)}
};
xData.setDataArray(aValues);

// insert a column label range
Object aLabelsObj = xDocProp.getPropertyValue("ColumnLabelRanges");
com.sun.star.sheet.XLabelRanges xLabelRanges = (com.sun.star.sheet.XLabelRanges)
    UnoRuntime.queryInterface(com.sun.star.sheet.XLabelRanges.class, aLabelsObj);
com.sun.star.table.CellRangeAddress aLabelArea = new com.sun.star.table.CellRangeAddress();
aLabelArea.Sheet       = 0;
aLabelArea.StartColumn = 6;
aLabelArea.StartRow    = 47;
aLabelArea.EndColumn   = 7;
aLabelArea.EndRow      = 47;
com.sun.star.table.CellRangeAddress aDataArea = new com.sun.star.table.CellRangeAddress();
aDataArea.Sheet       = 0;
aDataArea.StartColumn = 6;
aDataArea.StartRow    = 48;
aDataArea.EndColumn   = 7;
aDataArea.EndRow      = 49;
xLabelRanges.addNew(aLabelArea, aDataArea);

// use the label range in formulas
xSheet.getCellByPosition(8, 48).setFormula("=Apples+Oranges");
xSheet.getCellByPosition(8, 49).setFormula("=Apples+Oranges");
```

Querying for Cells with Specific Properties

Cells, cell ranges and collections of cell ranges are queried for certain cell contents through the service `com.sun.star.sheet.SheetRangesQuery`. It implements interfaces to query cells and cell ranges with specific properties.

The methods of the interface `com.sun.star.sheet.XCellRangesQuery` search for cells with specific contents or properties inside of the given cell range. The methods of the interface `com.sun.star.sheet.XFormulaQuery` search for cells in the entire spreadsheet that are reference to or are referenced from formula cells in the given range.

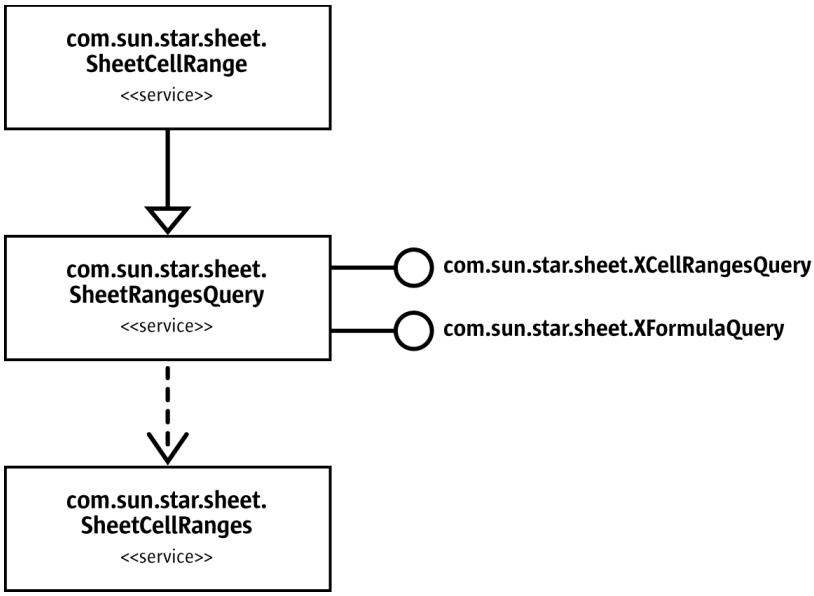


Illustration 104: Query sheet ranges

Due to a bug in the current implementation, both methods `queryPrecedents()` and `queryDependents()` of the interface `com.sun.star.sheet.XFormulaQuery` cause an endless loop in recursive mode, if parameter `bRecursive` is `true`.

All methods return the interface `com.sun.star.sheet.XSheetCellRanges` of a cell range collection. Cell range collections are described in the chapter *8.3.1 Spreadsheet Documents - Working with Spreadsheets - Document Structure - Cell Ranges and Cells Container.*

Methods of `com.sun.star.sheet.XCellRangesQuery`	
`queryVisibleCells()`	Returns all cells that are not hidden.
`queryEmptyCells()`	Returns all cells that do not have any content.
`queryContentCells()`	Returns all cells that have the contents described by the passed parameter. The flags are defined in `com.sun.star.sheet.CellFlags`.
`queryFormulaCells()`	Returns all formula cells whose results have a specific type described by the passed parameter. The result flags are defined in `com.sun.star.sheet.FormulaResult`.
`queryColumnDifferences()`	Returns all cells of the range that have different contents than the cell in the same column of the specified row. See the example below.
`queryRowDifferences()`	Returns all cells of the range that have different contents than the cell in the same row of the specified column. See the example below.
`queryIntersection()`	Returns all cells of the range that are contained in the passed range address.

Example:

	A	B	C	D	E	F	G
1	1	1	2				
2	1	2	2				
3	1	2	1				
4	1	1	1				

The queried range is A1:C4 and the passed cell address is B2.

- `queryColumnDifferences()`: (the row number is of interest) The cells of column A are compared with cell A2, the cells of column B with B2 and so on. The function returns the cell range list B1:B1, B4:B4, C3:C4.
- `queryRowDifferences()`: (the column index is of interest) The function compares row 1 with cell B1, row 2 with cell B2 and so on. It returns the cell range list C1:C1, A2:A2, A3:A3, C3:C3.

The following code queries all cells with text content: (Spreadsheet/SpreadsheetSample.java)

```
// --- Cell Ranges Query ---
// query addresses of all cells containing text
com.sun.star.sheet.XCellRangesQuery xRangesQuery = (com.sun.star.sheet.XCellRangesQuery)
    UnoRuntime.queryInterface(com.sun.star.sheet.XCellRangesQuery.class, xCellRange);

com.sun.star.sheet.XSheetCellRanges xCellRanges =
    xRangesQuery.queryContentCells((short)com.sun.star.sheet.CellFlags.STRING);
System.out.println("Cells containing text: " + xCellRanges.getRangeAddressesAsString());
```

Search and Replace

The cell range interface `com.sun.star.util.XReplaceable` is derived from `com.sun.star.util.XSearchable` providing search and replacement of text.

- The method `createReplaceDescriptor()` creates a new descriptor that contains all data for the replace action. It returns the interface `com.sun.star.util.XReplaceDescriptor` of this descriptor.
- The method `replaceAll()` performs a replacement in all cells according to the passed replace descriptor.

The following example replaces all occurrences of “cell” with “text”: (Spreadsheet/SpreadsheetSample.java)

```
// --- Replace text in all cells. ---
com.sun.star.util.XReplaceable xReplace = (com.sun.star.util.XReplaceable)
    UnoRuntime.queryInterface(com.sun.star.util.XReplaceable.class, xCellRange);
com.sun.star.util.XReplaceDescriptor xReplaceDesc = xReplace.createReplaceDescriptor();
xReplaceDesc.setSearchString("cell");
xReplaceDesc.setReplaceString("text");
// property SearchWords searches for whole cells!
xReplaceDesc.setPropertyValue("SearchWords", new Boolean(false));
int nCount = xReplace.replaceAll(xReplaceDesc);
System.out.println("Search text replaced " + nCount + " times.");
```



The property `SearchWords` has a different meaning in spreadsheets: If `true`, only cells containing the whole search text and nothing else is found. If `false`, cells containing the search string as a substring is found.

8.3.4 Sorting

Table Sort Descriptor

A *sort descriptor* describes all properties of a sort operation. The service `com.sun.star.table.TableSortDescriptor` extends the service `com.sun.star.util.SortDescriptor` with table specific sorting properties, such as:

The sorting orientation using the enumeration `com.sun.star.table.TableOrientation` property `Orientation`.

A sequence of sorting fields using the `SortFields` property that contains a sequence of `com.sun.star.util.SortField` structs.

The size of the sequence using the `MaxFieldCount` property.

The existence of column or row headers using the boolean property `ContainsHeader`.

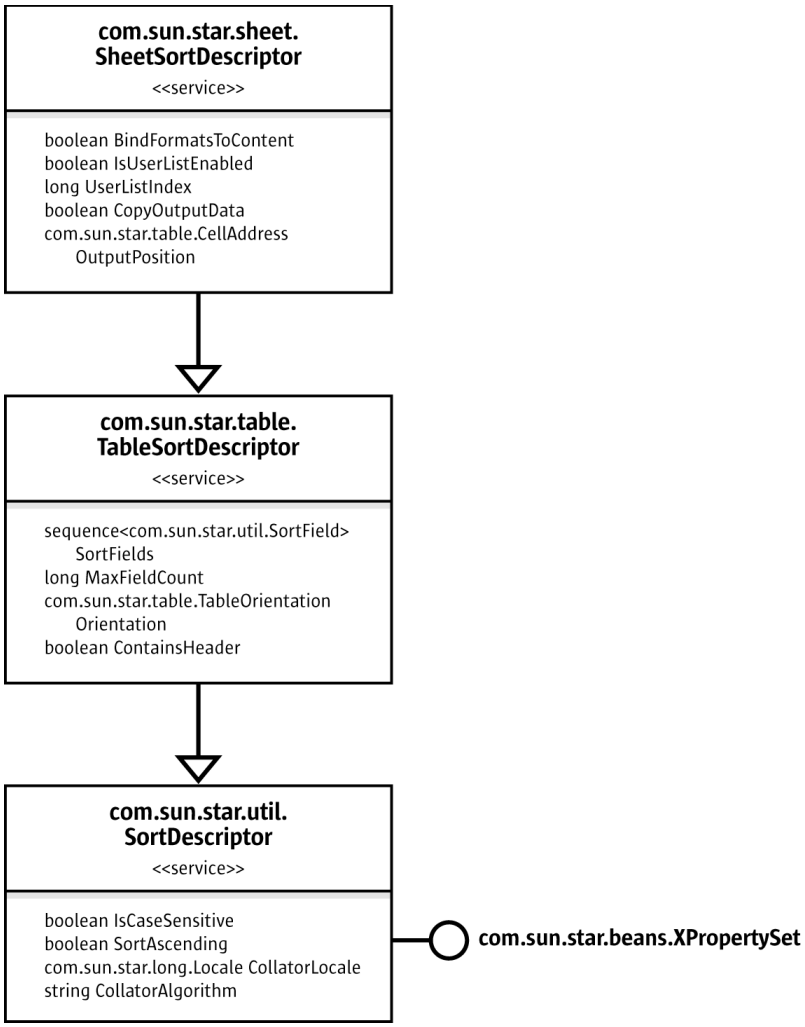


Illustration 105: SheetSortDescriptor

To sort the contents of a cell range, the `sort()` method from the `com.sun.star.util.XSortable` interface is called, passing a sequence of property values with properties from the `com.sun.star.sheet.SheetSortDescriptor` service. The sequence can be constructed from scratch containing the properties that should be set, or the return value of the `createSortDescriptor()` method can be used and modified. If the cell range is a database range that has a stored sort operation, `createSortDescriptor()` returns a sequence with the options of this sort operation.

The fields that the cell range is sorted by are specified in the `SortFields` property as a sequence of `com.sun.star.util.SortField` elements. In the `com.sun.star.util.SortField` struct, the `Field` member specifies the field number by which to sort, and the boolean `SortAscending` member switches between ascending and descending sorting for that field.



The `FieldType` member, that is used to select textual or numeric sorting in text documents is ignored in the spreadsheet application. In a spreadsheet, a cell always has a known type of text or value, which is used for sorting, with numbers sorted before text cells.

The `CopyOutputData` and `OutputPosition` properties are analogous to the filter descriptor's properties of the same name. The `SortAscending` property from the `com.sun.star.util.SortDescriptor` service is ignored, as the direction is selected for each field individually.

If the `IsUserListEnabled` property is `true`, a user-defined sort list is used that specifies an order for the strings it contains. The `UserListIndex` property selects an entry from the `UserLists` property of the `com.sun.star.sheet.GlobalSheetSettings` service to find the sort list that is used.

The `CollatorLocale` is used to sort according to the sorting rules of a given locale. For some locales, several different sorting rules exist. In this case, the `CollatorAlgorithm` is used to select one of the sorting rules. The `com.sun.star.i18n.Collator` service is used to find the possible `CollatorAlgorithm` values for a locale.

The following example sorts the cell range by the second column in ascending order: (Spreadsheet/SpreadsheetSample.java)

```
// --- sort by second column, ascending ---

// define the fields to sort
com.sun.star.util.SortField[] aSortFields = new com.sun.star.util.SortField[1];
aSortFields[0] = new com.sun.star.util.SortField();
aSortFields[0].Field         = 1;
aSortFields[0].SortAscending = true;

// define the sort descriptor
com.sun.star.beans.PropertyValue[] aSortDesc = new com.sun.star.beans.PropertyValue[2];
aSortDesc[0] = new com.sun.star.beans.PropertyValue();
aSortDesc[0].Name  = "SortFields";
aSortDesc[0].Value = aSortFields;
aSortDesc[1] = new com.sun.star.beans.PropertyValue();
aSortDesc[1].Name  = "ContainsHeader";
aSortDesc[1].Value = new Boolean(true);

// perform the sorting
com.sun.star.util.XSortable xSort = (com.sun.star.util.XSortable)
    UnoRuntime.queryInterface(com.sun.star.util.XSortable.class, xRange);
xSort.sort(aSortDesc);
```

8.3.5 Database Operations

This section discusses the operations that treat the contents of a cell range as database data, organized in rows and columns like a database table. These operations are filtering, sorting, adding of subtotals and importing from an external database. Each of the operations is controlled using a descriptor service. The descriptors can be used in two ways:

- Performing an operation on a cell range. This is described in the following sections about the individual descriptors.
- Accessing the settings that are stored with a database range. This is described in the section about database ranges.

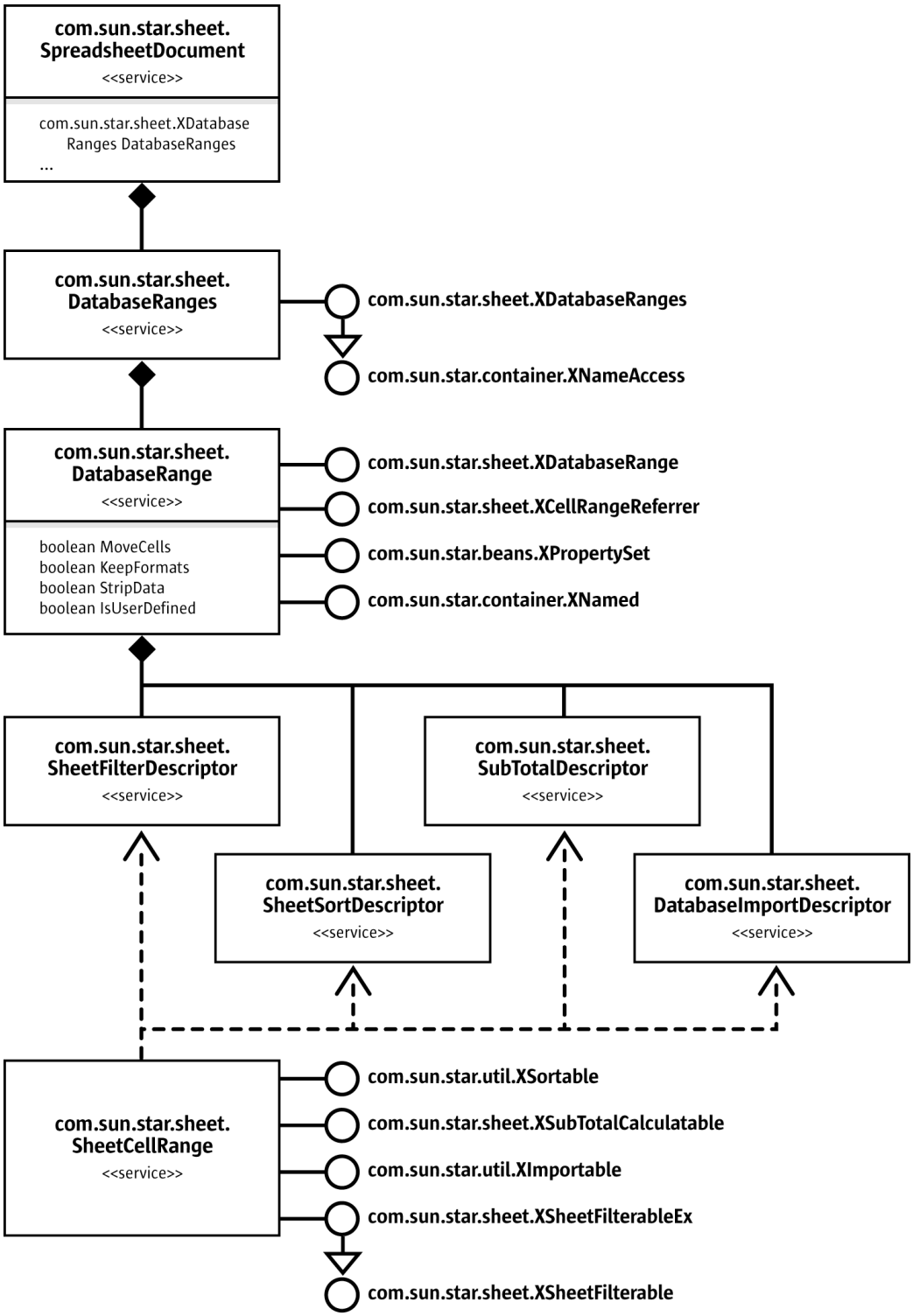


Illustration 106: DatabaseRange

Filtering

A `com.sun.star.sheet.SheetFilterDescriptor` object is created using the `createFilterDescriptor()` method from the range's `com.sun.star.sheet.XSheetFilterable` interface to filter

data in a cell range. After applying the settings to the descriptor, it is passed to the `filter()` method.

If `true` is passed as a `bEmpty` parameter to `createFilterDescriptor()`, the returned descriptor contains default values for all settings. If `false` is passed and the cell range is a database range that has a stored filter operation, the settings for that filter are used.

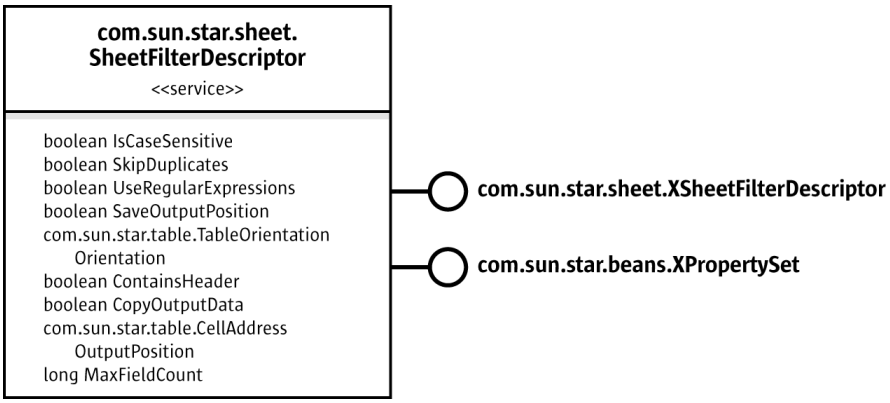


Illustration 107: SheetFilterDescriptor

The `com.sun.star.sheet.XSheetFilterDescriptor` interface is used to set the filter criteria as a sequence of `com.sun.star.sheet.TableFilterField` elements. The `com.sun.star.sheet.TableFilterField` struct describes a single condition and contains the following members:

- `Connection` has the values `AND` or `OR`, and specifies how the condition is connected to the previous condition in the sequence. For the first entry, `Connection` is ignored.
- `Field` is the number of the field that the condition is applied to.
- `Operator` is the type of the condition, such as `EQUAL` or `GREATER`
- `IsNumeric` selects a numeric or textual condition.
- `NumericValue` contains the value that is used in the condition if `IsNumeric` is `true`.
- `StringValue` contains the text that is used in the condition if `IsNumeric` is `false`.

Additionally, the filter descriptor contains a `com.sun.star.beans.XPropertySet` interface for settings that affect the whole filter operation.

If the property `CopyOutputData` is `true`, the data that matches the filter criteria is copied to a cell range in the document that starts at the position specified by the `OutputPosition` property. Otherwise, the rows that do not match the filter criteria are filtered (hidden) in the original cell range.

The following example filters the range that is in the variable `xRange` for values greater or equal to 1998 in the second column: (Spreadsheet/SpreadsheetSample.java)

```
// --- filter for second column >= 1998 ---
com.sun.star.sheet.XSheetFilterable xFilter = (com.sun.star.sheet.XSheetFilterable)
    UnoRuntime.queryInterface(com.sun.star.sheet.XSheetFilterable.class, xRange);
com.sun.star.sheet.XSheetFilterDescriptor xFilterDesc =
    xFilter.createFilterDescriptor(true);
com.sun.star.sheet.TableFilterField[] aFilterFields =
    new com.sun.star.sheet.TableFilterField[1];
aFilterFields[0] = new com.sun.star.sheet.TableFilterField();
aFilterFields[0].Field        = 1;
aFilterFields[0].IsNumeric    = true;
aFilterFields[0].Operator     = com.sun.star.sheet.FilterOperator.GREATER_EQUAL;
aFilterFields[0].NumericValue = 1998;
xFilterDesc.setFilterFields(aFilterFields);
com.sun.star.beans.XPropertySet xFilterProp = (com.sun.star.beans.XPropertySet)
    UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, xFilterDesc);
xFilterProp.setPropertyValue("ContainsHeader", new Boolean(true));
xFilter.filter(xFilterDesc);
```

The `com.sun.star.sheet.XSheetFilterableEx` interface is used to create a filter descriptor from criteria in a cell range in the same manner as the “Advanced Filter” dialog. The `com.sun.star.sheet.XSheetFilterableEx` interface must be queried from the range that contains the conditions, and the `com.sun.star.sheet.XSheetFilterable` interface of the range to be filtered must be passed to the `createFilterDescriptorByObject()` call.

The following example performs the same filter operation as the example before, but reads the filter criteria from a cell range:

```
// --- do the same filter as above, using criteria from a cell range ---
com.sun.star.table.XCellRange xCritRange = xSheet.getCellRangeByName("B27:B28");
com.sun.star.sheet.XCellRangeData xCritData = (com.sun.star.sheet.XCellRangeData)
    UnoRuntime.queryInterface(com.sun.star.sheet.XCellRangeData.class, xCritRange);
Object[][] aCritValues = {{"Year"}, {">= 1998"}};
xCritData.setDataArray(aCritValues);
com.sun.star.sheet.XSheetFilterableEx xCriteria = (com.sun.star.sheet.XSheetFilterableEx)
    UnoRuntime.queryInterface(com.sun.star.sheet.XSheetFilterableEx.class, xCritRange);
xFilterDesc = xCriteria.createFilterDescriptorByObject(xFilter);
if (xFilterDesc != null)
    xFilter.filter(xFilterDesc);
```

Subtotals

A `com.sun.star.sheet.SubTotalDescriptor` object is created using the `createSubTotalDescriptor()` method from the range's `com.sun.star.sheet.XSubTotalCalculatable` interface to create subtotals for a cell range. After applying the settings to the descriptor, it is passed to the `applySubTotals()` method.

The `bEmpty` parameter to the `createSubTotalDescriptor()` method works in the same manner as the parameter to the `createFilterDescriptor()` method described in the filtering section. If the `bReplace` parameter to the `applySubTotals()` method is `true`, existing subtotal rows are deleted before inserting new ones.

The `removeSubTotals()` method removes the subtotal rows from the cell range without modifying the stored subtotal settings, so that the same subtotals can later be restored.

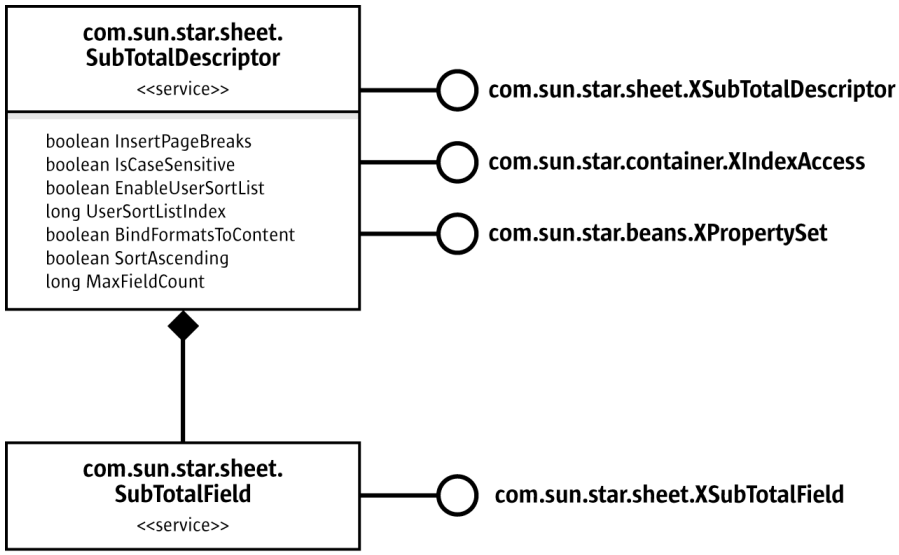


Illustration 108: SubtotalDescriptor

New fields are added to the subtotal descriptor using the `com.sun.star.sheet.XSubTotalDescriptor` interface's `addNew()` method. The `nGroupColumn` parameter selects the column by which values are grouped. The subtotals are inserted at changes of the column's values. The `aSubTotalColumns` parameter specifies which column subtotal values are calculated. It is a sequence of `com.sun.star.sheet.SubTotalColumn` entries where each entry contains the column number and the function to be calculated.

To query or modify the fields in a subtotal descriptor, the `com.sun.star.container.XIndexAccess` interface is used to access the fields. Each field's `com.sun.star.sheet.XSubTotalField` interface gets and sets the group and subtotal columns.

The example below creates subtotals, grouping by the first column and calculating the sum of the third column: (Spreadsheet/SpreadsheetSample.java)

```
// --- insert subtotals ---
com.sun.star.sheet.XSubTotalCalculatable xSub = (com.sun.star.sheet.XSubTotalCalculatable)
    UnoRuntime.queryInterface(com.sun.star.sheet.XSubTotalCalculatable.class, xRange);
com.sun.star.sheet.XSubTotalDescriptor xSubDesc = xSub.createSubTotalDescriptor(true);
com.sun.star.sheet.SubTotalColumn[] aColumns = new com.sun.star.sheet.SubTotalColumn[1];
// calculate sum of third column
aColumns[0] = new com.sun.star.sheet.SubTotalColumn();
aColumns[0].Column   = 2;
aColumns[0].Function = com.sun.star.sheet.GeneralFunction.SUM;
// group by first column
xSubDesc.addNew(aColumns, 0);
xSub.applySubTotals(xSubDesc, true);
```

Database Import

The `com.sun.star.util.XImportable` interface imports data from an external data source (database) into spreadsheet cells. The database has to be registered in OpenOffice.org API, so that it can

be selected using its name. The `doImport` call takes a sequence of property values that select the data to import.

Similar to the sort descriptor, the import descriptor's sequence of property values can be constructed from scratch, or the return value of the `createImportDescriptor()` method can be used and modified. The `createImportDescriptor()` method returns a description of the previously imported data if the cell range is a database range with stored import settings and the `bEmpty` parameter is `false`.

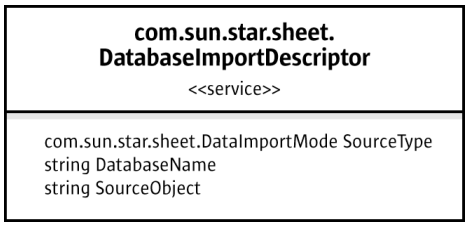


Illustration 109:
DatabaseImportDescriptor

The `DatabaseName` property selects a database. The `SourceType` selects the kind of object from the database that is imported. It can have the following values:

- If `SourceType` is `TABLE`, the whole table that is named by `SourceObject` is imported.
- If `SourceType` is `QUERY`, the `SourceObject` must be the name of a named query.
- If `SourceType` is `SQL`, the `SourceObject` is used as a literal SQL command string.

If a database name is in the `aDatabase` variable and a table name in `aTableName`, the following code imports that table from the database: (Spreadsheet/SpreadsheetSample.java)

```
// --- import from database ---
com.sun.star.beans.PropertyValue[] aImportDesc = new com.sun.star.beans.PropertyValue[3];
aImportDesc[0] = new com.sun.star.beans.PropertyValue();
aImportDesc[0].Name  = "DatabaseName";
aImportDesc[0].Value = aDatabase;
aImportDesc[1] = new com.sun.star.beans.PropertyValue();
aImportDesc[1].Name  = "SourceType";
aImportDesc[1].Value = com.sun.star.sheet.DataImportMode.TABLE;
aImportDesc[2] = new com.sun.star.beans.PropertyValue();
aImportDesc[2].Name  = "SourceObject";
aImportDesc[2].Value = aTableName;
com.sun.star.table.XCellRange xImportRange = xSheet.getCellRangeByName("B33:B33");
com.sun.star.util.XImportable xImport = ( com.sun.star.util.XImportable )
    UnoRuntime.queryInterface(com.sun.star.util.XImportable.class, xImportRange);
xImport.doImport(aImportDesc);
```

Database Ranges

A database range is a name for a cell range that also stores filtering, sorting, subtotal and import settings, as well as some options.

The `com.sun.star.sheet.SpreadsheetDocument` service has a property `DatabaseRanges` that is used to get the document's collection of database ranges. A new database range is added using the `com.sun.star.sheet.XDatabaseRanges` interface's `addNewByName()` method that requires the name of the new database range, and a `com.sun.star.table.CellRangeAddress` with the address of the cell range as arguments. The `removeByName()` method removes a database range.

The `com.sun.star.container.XNameAccess` interface is used to get a single `com.sun.star.sheet.DatabaseRange` object. Its `com.sun.star.sheet.XCellRangeReferrer` interface is used to access the cell range that it is pointed to. The `com.sun.star.sheet.XDatabaseRange` interface retrieves or changes the `com.sun.star.table.CellRangeAddress` that is named, and gets the stored descriptors.

All descriptors of a database range are updated when a database operation is carried out on the cell range that the database range points to. The stored filter descriptor and subtotal descriptor can also be modified by changing the objects that are returned by the `getFilterDescriptor()` and `getSubTotalDescriptor()` methods. Calling the `refresh()` method carries out the stored operations again.

Whenever a database operation is carried out on a cell range where a database range is not defined, a temporary database range is used to hold the settings. This temporary database range has its `IsUserDefined` property set to `false` and is valid until another database operation is performed on a different cell range. In this case, the temporary database range is modified to refer to the new cell range.

The following example uses the `IsUserDefined` property to find the temporary database range, and applies a background color to the corresponding cell range. If run directly after the database import example above, this marks the imported data. (Spreadsheet/SpreadsheetSample.java)

```
// use the temporary database range to find the imported data's size
com.sun.star.beans.XPropertySet xDocProp = (com.sun.star.beans.XPropertySet)
    UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, getDocument());
Object aRangesObj = xDocProp.getPropertyValue("DatabaseRanges");
com.sun.star.container.XNameAccess xRanges = (com.sun.star.container.XNameAccess)
    UnoRuntime.queryInterface(com.sun.star.container.XNameAccess.class, aRangesObj);
String[] aNames = xRanges.getElementNames();
for (int i=0; i<aNames.length; i++) {
    Object aRangeObj = xRanges.getByName(aNames[i] );
    com.sun.star.beans.XPropertySet xRangeProp = (com.sun.star.beans.XPropertySet)
        UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, aRangeObj);
    boolean bUser = ((Boolean) xRangeProp.getPropertyValue("IsUserDefined")).booleanValue();
    if (!bUser) {
        // this is the temporary database range - get the cell range and format it
        com.sun.star.sheet.XCellRangeReferrer xRef = (com.sun.star.sheet.XCellRangeReferrer)
            UnoRuntime.queryInterface(com.sun.star.sheet.XCellRangeReferrer.class, aRangeObj);
        com.sun.star.table.XCellRange xResultRange = xRef.getReferredCells();
        com.sun.star.beans.XPropertySet xResultProp = (com.sun.star.beans.XPropertySet)
            UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, xResultRange);
        xResultProp.setPropertyValue("IsCellBackgroundTransparent", new Boolean(false));
        xResultProp.setPropertyValue("CellBackColor", new Integer(0xFFFFCC));
    }
}
```

8.3.6 Linking External Data

This section explains different ways to link data from external sources into a spreadsheet document. Refer to the *8.3.5 Spreadsheet Documents - Working with Spreadsheets - Database Operations - Database Import* chapter for linking data from a database.

Sheet Links

Each sheet in a spreadsheet document can be linked to a sheet from a different document. The spreadsheet document has a collection of all the sheet links to different source documents.

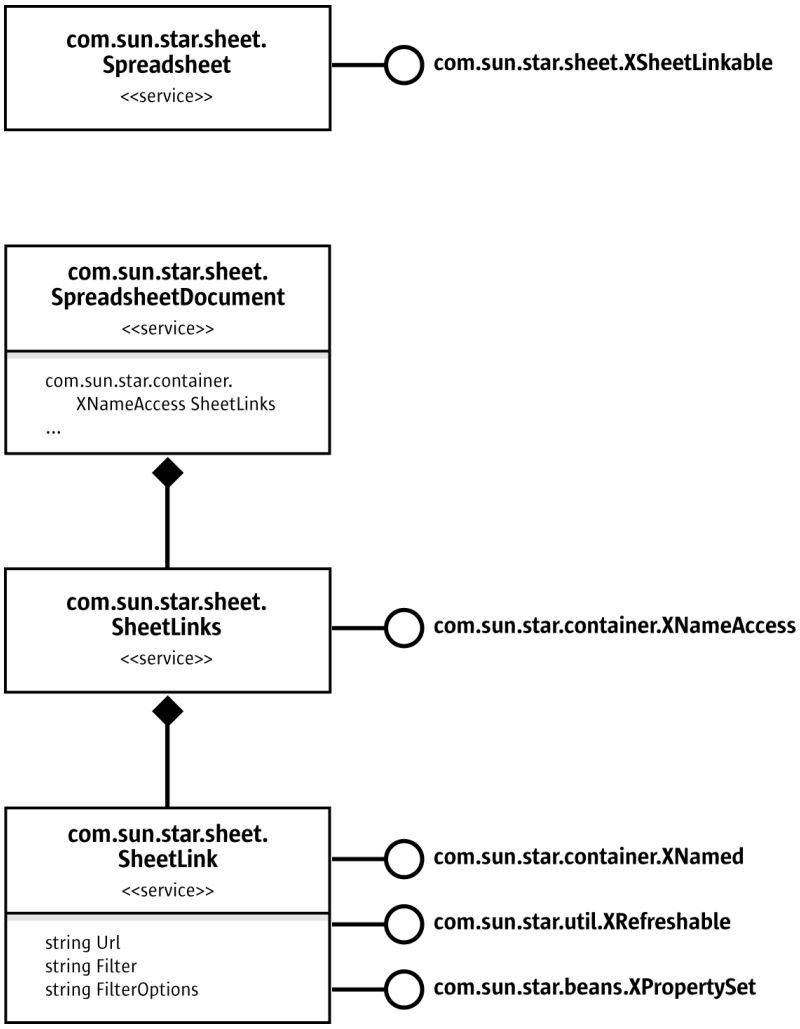


Illustration 110: SheetLinks

The interface `com.sun.star.sheet.XSheetLinkable` is relevant if the current sheet is used as buffer for an external sheet link. The interfaces provides access to the data of the link. A link is established using the `com.sun.star.sheet.XSheetLinkable` interface's `link()` method. The method's parameters are:

- The source document's URL. When a sheet link is inserted or updated, the source document is loaded from its URL. Unsaved changes in a source document that is open in memory are not included. All URL types that can be used to load files can also be used in links, including HTTP to link to data from a web server.
- The name of the sheet in the source document from the contents are copied from. If this string is empty, the source document's first sheet is used, regardless of its name.
- The filter name and options that are used to load the source document. Refer to the *6.1.5 Office Development - OpenOffice.org Application Environment - Handling Documents* chapter. All spreadsheet file filters can be used, so it is possible, for example, to link to a CSV text file.
- A `com.sun.star.sheet.SheetLinkMode` enum value that controls how the contents are copied:
  - If the mode is `NORMAL`, all cells from the source sheet are copied, including formulas.
  - If the mode is `VALUE`, formulas are replaced by their results in the copy.

The link mode, source URL and source sheet name can also be queried and changed using the `getLinkMode()`, `setLinkMode()`, `getLinkUrl()`, `setLinkUrl()`, `getLinkSheetName()` and `setLinkSheetName()` methods. Setting the mode to `NONE` removes the link.

The `com.sun.star.sheet.SheetLinks` collection contains an entry for every source document that is used in sheet links. If several sheets are linked to different sheets from the same source document, there is only one entry for them. The name that is used for the `com.sun.star.container.XNameAccess` interface is the source document's URL.

The `com.sun.star.sheet.SheetLink` service changes a link's source URL, filter or filter options through the `com.sun.star.beans.XPropertySet` interface. The `com.sun.star.util.XRefreshable` interface is used to update the link. This affects all sheets that are linked to any sheet from the link's source document.



External references in cell formulas are implemented using hidden linked sheets that show as sheet link objects.

Cell Area Links

A cell area link is a cell area (range) in a spreadsheet that is linked to a cell area from a different document.

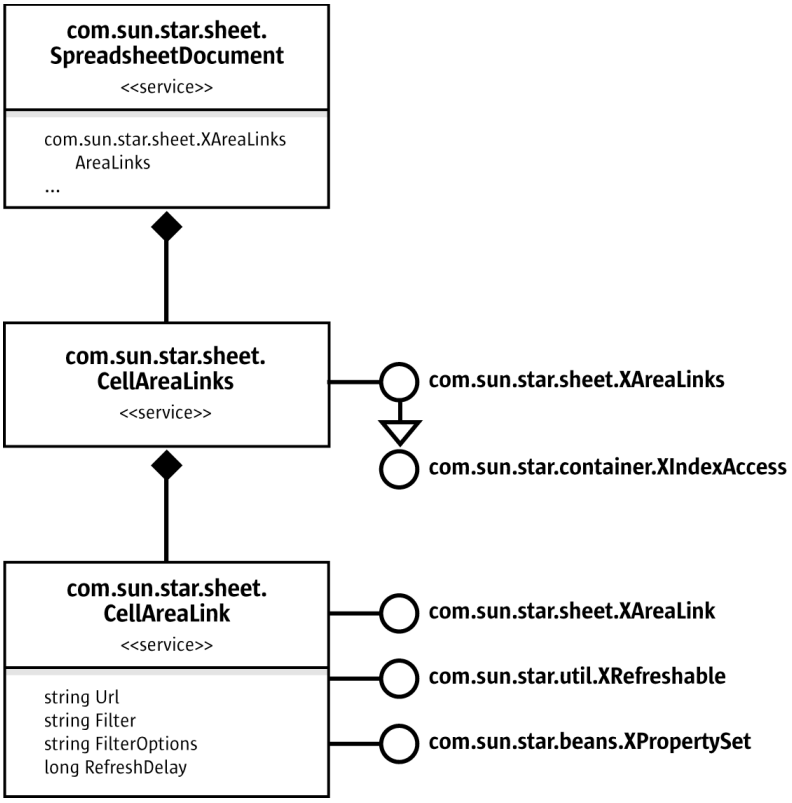


Illustration 111: CellAreaLinks

To insert an area link, the `com.sun.star.sheet.XAreaLinks` interface's `insertAtPosition()` method is used with the following parameters:

- The position where the link is placed in the document as a `com.sun.star.table.CellAddress` struct.

- The source document's URL is used in the same manner as sheet links.
- A string describing the source range in the source document. This can be the name of a named range or database range, or a direct cell reference, such as “sheet1.a1:c5”. Note that the WebQuery import filter creates a named range for each HTML table. These names can be used also.
- The filter name and filter options are used in the same manner as sheet links.

The `removeByIndex()` method is used to remove a link.

The `com.sun.star.sheet.CellAreaLink` service is used to modify or refresh an area link. The `com.sun.star.sheet.XAreaLink` interface queries and modifies the link's source range and its output range in the document. Note that the output range changes in size after updating if the size of the source range changes.

The `com.sun.star.beans.XPropertySet` interface changes the link's source URL, filter name and filter options. Unlike sheet links, these changes affect only one linked area. Additionally, the `RefreshDelay` property is used to set an interval in seconds to periodically update the link. If the value is 0, no automatic updates occur.

The `com.sun.star.util.XRefreshable` interface is used to update the link.

DDE Links

A DDE link is created whenever the DDE spreadsheet function is used in a cell formula.

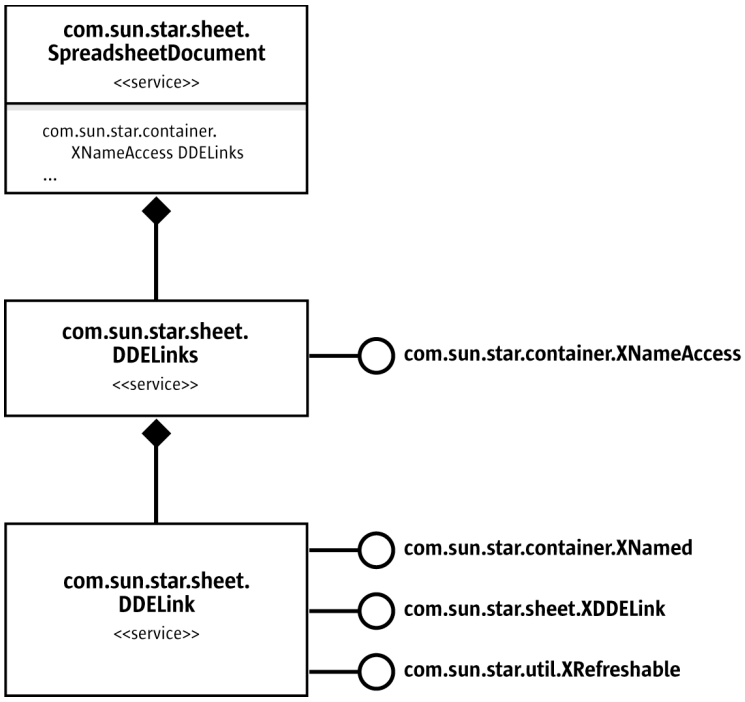


Illustration 112: DDELink

The `com.sun.star.sheet.DDELink` service is only used to query the link's parameters using the `com.sun.star.sheet.XDDELink` interface, and refresh it using the `com.sun.star.util.XRefreshable` interface. The DDE link's parameters, *Application*, *Topic* and

Item are determined by the formula that contains the DDE function, therefore it is not possible to change these parameters in the link object.

The link's name used for the `com.sun.star.container.XNameAccess` interface consists of the three parameter strings concatenated.

8.3.7 DataPilot

DataPilot Tables

The `com.sun.star.sheet.DataPilotTables` and related services create and modify DataPilot tables in a spreadsheet.

The method `getDataPilotTables()` of the interface `com.sun.star.sheet.XDataPilotTablesSupplier` returns the interface `com.sun.star.sheet.XDataPilotTables` of the collection of all data pilot tables contained in the spreadsheet.

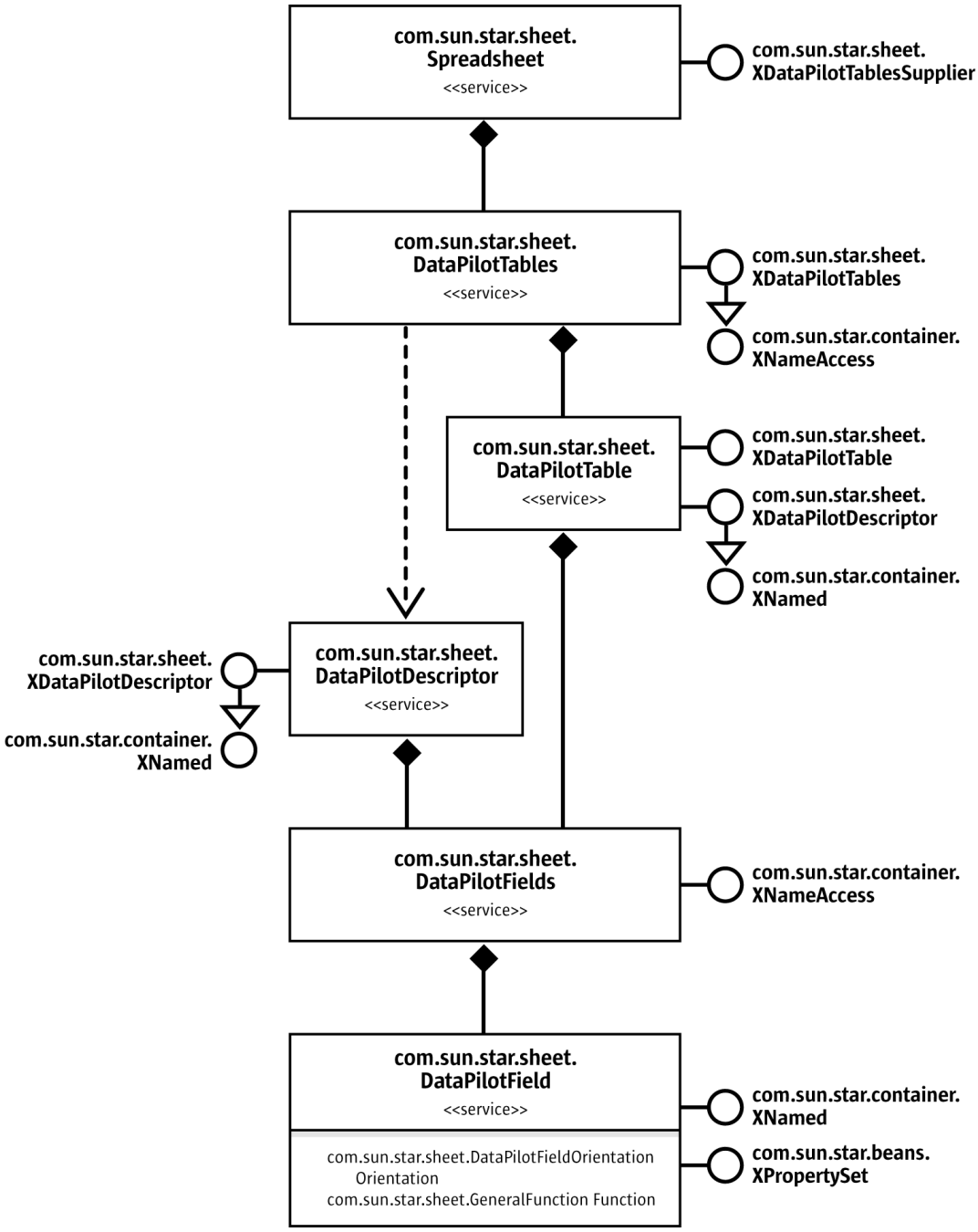


Illustration 113: DataPilotTables

The `com.sun.star.sheet.DataPilotTables` service is accessed by getting the `com.sun.star.sheet.XDataPilotTablesSupplier` interface from a spreadsheet object and calling the `getDataPilotTables()` method.



Only DataPilot tables that are based on cell data are supported by these services. DataPilot tables created directly from external data sources or using the `com.sun.star.sheet.DataPilotSource` service cannot be created or modified this way.

Creating a New DataPilot Table

The first step to creating a new DataPilot table is to create a new `com.sun.star.sheet.DataPilotDescriptor` object by calling the `com.sun.star.sheet.XDataPilotTables` interface's `createDataPilotDescriptor()` method. The descriptor is then used to describe the DataPilot table's layout and options, and passed to the `insertNewByName()` method of `XDataPilotTables`. The other parameters for `insertNewByName()` are the name for the new table, and the position where the table is to be placed on the spreadsheet.

The `com.sun.star.sheet.XDataPilotDescriptor` interface offers methods to change the DataPilot table settings:

- The cell range that contains the source data is set with the `setSourceRange()` method. It is a `com.sun.star.table.CellRangeAddress` struct.
- The individual fields are handled using the `getDataPilotFields()`, `getColumnFields()`, `getRowFields()`, `getPageFields()`, `getDataFields()` and `getHiddenFields()` methods. The details are discussed below.
- The `setTag()` method sets an additional string that is stored with the DataPilot table, but does not influence its results.
- The `getFilterDescriptor()` method returns a `com.sun.star.sheet.SheetFilterDescriptor` object that can be used to apply filter criteria to the source data. Refer to the section on data operations for details on how to use a filter descriptor.

The layout of the DataPilot table is controlled using the `com.sun.star.sheet.DataPilotFields` service. Each `com.sun.star.sheet.DataPilotField` object has a property `Orientation` that controls where in the DataPilot table the field is used. The `com.sun.star.sheet.DataPilotFieldOrientation` enum contains the possible orientations:

- `HIDDEN`: The field is not used in the table.
- `COLUMN`: Values from this field are used to determine the columns of the table.
- `ROW`: Values from this field are used to determine the rows of the table.
- `PAGE`: The field is used in the table's “page” area, where single values from the field can be selected.
- `DATA`: The values from this field are used to calculate the table's data area.

The `Function` property is used to assign a function to the field. For instance, if the field has a `DATA` orientation, this is the function that is used for calculation of the results. If the field has `COLUMN` or `ROW` orientation, it is the function that is used to calculate subtotals for the values from this field.

The `getDataPilotFields()` method returns a collection containing one `com.sun.star.sheet.DataPilotField` entry for each column of source data, and one additional entry for the “Data” column that becomes visible when two or more fields get the `DATA` orientation. Each source column appears only once, even if it is used with several orientations or functions.

The `getColumnFields()`, `getRowFields()`, `getPageFields()` and `getDataFields()` methods each return a collection of the fields with the respective orientation. In the case of `getDataFields()`, a single source column can appear several times if it is used with different functions. The `getHiddenFields()` method returns a collection of those fields from the `getDataPilotFields()` collection that are not in any of the other collections.



Note: Page fields and the PAGE orientation are not supported by the current implementation. Setting a field's orientation to PAGE has the same effect as using HIDDEN. The getPageFields() method always returns an empty collection.

The exact effect of changing a field orientation depends on which field collection the field object was taken from. If the object is from the getDataPilotFields() collection, the field is added to the collection that corresponds to the new Orientation value. If the object is from any of the other collections, the field is removed from the old orientation and added to the new orientation.

The following example creates a simple DataPilot table with one column, row and data field. (Spreadsheet/SpreadsheetSample.java)

```
// --- Create a new DataPilot table ---
com.sun.star.sheet.XDataPilotTablesSupplier xDPSupp = (com.sun.star.sheet.XDataPilotTablesSupplier)
    UnoRuntime.queryInterface(com.sun.star.sheet.XDataPilotTablesSupplier.class, xSheet);
com.sun.star.sheet.XDataPilotTables xDPTables = xDPSupp.getDataPilotTables();
com.sun.star.sheet.XDataPilotDescriptor xDPDesc = xDPTables.createDataPilotDescriptor();

// set source range (use data range from CellRange test)
com.sun.star.table.CellRangeAddress aSourceAddress = createCellRangeAddress(xSheet, "A10:C30");
xDPDesc.setSourceRange(aSourceAddress);

// settings for fields
com.sun.star.container.XIndexAccess xFields = xDPDesc.getDataPilotFields();
Object aFieldObj;
com.sun.star.beans.XPropertySet xFieldProp;

// use first column as column field
aFieldObj = xFields.getByIndex(0);
xFieldProp = (com.sun.star.beans.XPropertySet)
    UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, aFieldObj);
xFieldProp.setPropertyValue("Orientation", com.sun.star.sheet.DataPilotFieldOrientation.COLUMN);

// use second column as row field
aFieldObj = xFields.getByIndex(1);
xFieldProp = (com.sun.star.beans.XPropertySet)
    UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, aFieldObj);
xFieldProp.setPropertyValue("Orientation", com.sun.star.sheet.DataPilotFieldOrientation.ROW);

// use third column as data field, calculating the sum
aFieldObj = xFields.getByIndex(2);
xFieldProp = (com.sun.star.beans.XPropertySet)
    UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, aFieldObj);
xFieldProp.setPropertyValue("Orientation", com.sun.star.sheet.DataPilotFieldOrientation.DATA);
xFieldProp.setPropertyValue("Function", com.sun.star.sheet.GeneralFunction.SUM);

// select output position
com.sun.star.table.CellAddress aDestAddress = createCellAddress(xSheet, "A40");
xDPTables.insertNewByName("DataPilotExample", aDestAddress, xDPDesc);
```

Modifying a DataPilot Table

The com.sun.star.sheet.DataPilotTable service is used to modify an existing DataPilot table. The object for an existing table is available through the com.sun.star.container.XNameAccess interface of the com.sun.star.sheet.DataPilotTables service. It implements the com.sun.star.sheet.XDataPilotDescriptor interface, so that the DataPilot table can be modified in the same manner as the descriptor for a new table in the preceding section. After any change to a DataPilot table's settings, the table is automatically recalculated.

Additionally, the com.sun.star.sheet.XDataPilotTable interface offers a getOutputRange() method that is used to find which range on the spreadsheet the table occupies, and a refresh() method that recalculates the table without changing any settings.

The following example modifies the table from the previous example to contain a second data field using the same source column as the existing data field, but using the “average” function instead. (Spreadsheet/SpreadsheetSample.java)

```
// --- Modify the DataPilot table ---
Object aDPTableObj = xDPTables.getByName("DataPilotExample");
xDPDesc = (com.sun.star.sheet.XDataPilotDescriptor)
    UnoRuntime.queryInterface(com.sun.star.sheet.XDataPilotDescriptor.class, aDPTableObj);
xFields = xDPDesc.getDataPilotFields();

// add a second data field from the third column, calculating the average
aFieldObj = xFields.getByIndex(2);
xFieldProp = (com.sun.star.beans.XPropertySet)
    UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, aFieldObj);
xFieldProp.setPropertyValue("Orientation", com.sun.star.sheet.DataPilotFieldOrientation.DATA);
xFieldProp.setPropertyValue("Function", com.sun.star.sheet.GeneralFunction.AVERAGE);
```



Note how the field object for the third column is taken from the collection returned by `getDataPilotFields()` to create a second data field. If the field object was taken from the collection returned by `getDataFields()`, only the existing data field's function would be changed by the `setPropertyValue()` calls to that object.

Removing a DataPilot Table

To remove a DataPilot table from a spreadsheet, call the `com.sun.star.sheet.XDataPilotTables` interface's `removeByName()` method, passing the DataPilot table's name.

DataPilot Sources

The DataPilot feature in OpenOffice.org API Calc makes use of an external component that provides the tabular results in the DataPilot table using the field orientations and other settings that are made in the DataPilot dialog or interactively by dragging the fields in the spreadsheet.

Such a component might, for example, connect to an OLAP server, allowing the use of a DataPilot table to interactively display results from that server.

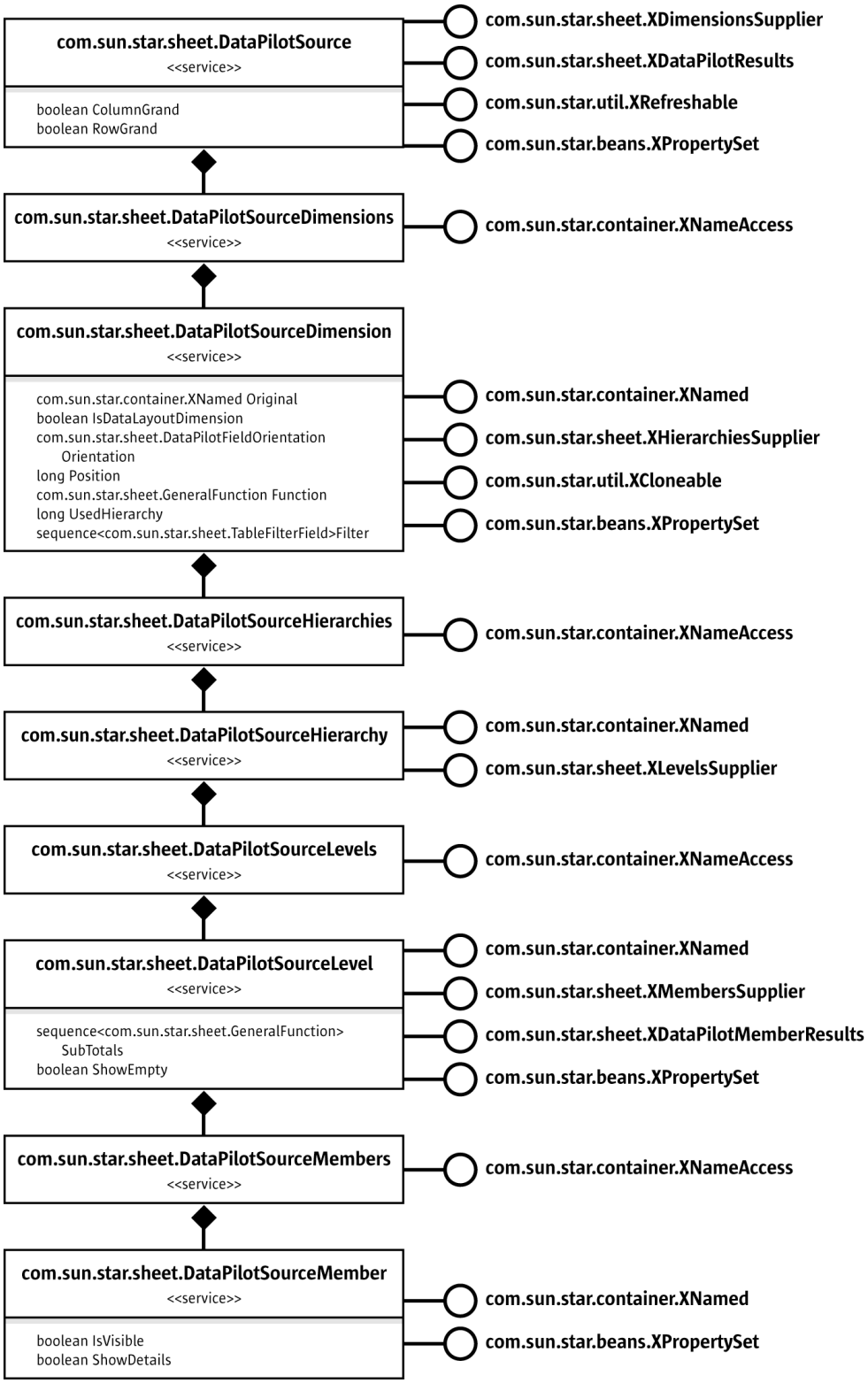


Illustration 114: DataPilotSource

The example that is used here provides four dimensions with the same number of members each, and one data dimension that uses these members as digits to form integer numbers. A resulting DataPilot table look similar to the following:

		hundreds		
ones	tens	0	1	2
0	0	0	100	200
	1	10	110	210
	2	20	120	220
1	0	1	101	201
	1	11	111	211
	2	21	121	221
2	0	2	102	202
	1	12	112	212
	2	22	122	222

The example uses the following class to hold the settings that are applied to the DataPilot source: (Spreadsheet/ExampleDataPiloSource.java)

```
class ExampleSettings
{
    static public final int nDimensionCount = 6;
    static public final int nValueDimension = 4;
    static public final int nDataDimension = 5;
    static public final String [] aDimensionNames = {
        "ones", "tens", "hundreds", "thousands", "value", "" };

    static public final String getMemberName(int nMember) {
        return String.valueOf(nMember);
    }

    public int nMemberCount = 3;
    public java.util.List aColDimensions = new java.util.ArrayList();
    public java.util.List aRowDimensions = new java.util.ArrayList();
}
```

To create a DataPilot table using a DataPilot source component, three steps are carried out:

1. The application gets the list of available dimensions (fields) from the component.
2. The application applies the user-specified settings to the component.
3. The application gets the results from the component.

The same set of objects are used for all three steps. The root object from which the other objects are accessed is the implementation of the `com.sun.star.sheet.DataPilotSource` service.

The `com.sun.star.sheet.DataPilotSourceDimensions`, `com.sun.star.sheet.DataPilotSourceHierarchies`, `com.sun.star.sheet.DataPilotSourceLevels` and `com.sun.star.sheet.DataPilotSourceMembers` services are accessed using their parent object interfaces. That is:

- `com.sun.star.sheet.DataPilotSourceDimensions` is the parent object of `com.sun.star.sheet.XDimensionsSupplier`
- `com.sun.star.sheet.DataPilotSourceHierarchies` is the parent object of `com.sun.star.sheet.XHierarchiesSupplier`
- `com.sun.star.sheet.DataPilotSourceLevels` is the parent object of `com.sun.star.sheet.XLevelsSupplier`
- `com.sun.star.sheet.DataPilotSourceMembers` is the parent object of `com.sun.star.sheet.XMembersSupplier`

All contain the `com.sun.star.container.XNameAccess` interface to access their children.

Source Object

An implementation of the `com.sun.star.sheet.DataPilotSource` service must be registered, so that a component can be used as a DataPilot source. If any implementations for the service are present, the **External source/interface** option in the DataPilot **Select Source** dialog is enabled. Any of the implementations can then be selected by its implementation name in the **External Source** dialog, along with four option strings labeled “Source”, “Name”, “User” and “Password”. The four options are passed to the component unchanged.

The option strings are passed to the `com.sun.star.lang.XInitialization` interface's `initialize()` method if that interface is present. The sequence that is passed to the call contains four strings with the values from the dialog. Note that the “Password” string is only saved in OpenOffice.org API's old binary file format, but not in the XML-based format. If the component needs a password, for example, to connect to a database, it must be able to prompt for that password.

The example below uses the first of the strings to determine how many members each dimension should have: (Spreadsheet/ExampleDataPiloSource.java)

```
private ExampleSettings aSettings = new ExampleSettings();

public void initialize(Object[] aArguments) {
    //  If the first argument (Source) is a number between 2 and 10,
    //  use it as member count, otherwise keep the default value.
    if (aArguments.length >= 1) {
        String aSource = (String) aArguments[0];
        if (aSource != null) {
            try {
                int nValue = Integer.parseInt(aSource);
                if (nValue >= 2 && nValue <= 10)
                    aSettings.nMemberCount = nValue;
            } catch (NumberFormatException e) {
            }
        }
    }
}
```

The source object's `com.sun.star.beans.XPropertySet` interface is used to apply two settings: The `ColumnGrand` and `RowGrand` properties control if grand totals for columns or rows should be added. The settings are taken from the **DataPilot** dialog. The example does not use them.

The `com.sun.star.sheet.XDataPilotResults` interface is used to query the results from the component. This includes only the numeric “data” part of the table. In the example table above, it would be the 9x3 area of cells that are right-aligned. The `getResults()` call returns a sequence of rows, where each row is a sequence of the results for that row. The `com.sun.star.sheet.DataResult` struct contains the numeric value in the `Value` member, and a `Flags` member contains a combination of the `com.sun.star.sheet.DataResultFlags` constants:

- `HASDATA` is set if there is a valid result at the entry's position. A result value of zero is different from no result, so this must be set only if the result is not empty.
- `SUBTOTAL` marks a subtotal value that is formatted differently in the DataPilot table output.
- `ERROR` is set if the result at the entry's position is an error.

In the example table above, all entries have different `Value` numbers, and a `Flags` value of `HASDATA`. The implementation for the example looks like this:
(Spreadsheet/ExampleDataPiloSource.java)

```
public com.sun.star.sheet.DataResult[][] getResults() {
    int[] nDigits = new int[ExampleSettings.nDimensionCount];
    int nValue = 1;
    for (int i=0; i<ExampleSettings.nDimensionCount; i++) {
        nDigits[i] = nValue;
        nValue *= 10;
    }

    int nMemberCount = aSettings.nMemberCount;
    int nRowDimCount = aSettings.aRowDimensions.size();
    int nColDimCount = aSettings.aColDimensions.size();

    int nRows = 1;
    for (int i=0; i<nRowDimCount; i++)
        nRows *= nMemberCount;
    int nColumns = 1;
    for (int i=0; i<nColDimCount; i++)
        nColumns *= nMemberCount;

    com.sun.star.sheet.DataResult[][] aResults = new com.sun.star.sheet.DataResult[nRows][];
    for (int nRow=0; nRow<nRows; nRow++) {
        int nRowVal = nRow;
        int nRowResult = 0;
        for (int nRowDim=0; nRowDim<nRowDimCount; nRowDim++) {
            int nDim = ((Integer)aSettings.aRowDimensions.get(nRowDimCount-nRowDim-1)).intValue();
            nRowResult += ( nRowVal % nMemberCount ) * nDigits[nDim];
            nRowVal /= nMemberCount;
        }

        aResults[nRow] = new com.sun.star.sheet.DataResult[nColumns];
        for (int nCol=0; nCol<nColumns; nCol++) {
            int nColVal = nCol;
            int nResult = nRowResult;
            for (int nColDim=0; nColDim<nColDimCount; nColDim++) {
                int nDim = ((Integer)
                    aSettings.aColDimensions.get(nColDimCount-nColDim-1)).intValue();
                nResult += (nColVal % nMemberCount) * nDigits[nDim];
                nColVal /= nMemberCount;
            }

            aResults[nRow][nCol] = new com.sun.star.sheet.DataResult();
            aResults[nRow][nCol].Flags = com.sun.star.sheet.DataResultFlags.HASDATA;
            aResults[nRow][nCol].Value = nResult;
        }
    }
    return aResults;
}
```

The `com.sun.star.util.XRefreshable` interface contains a `refresh()` method that tells the component to discard cached results and recalculate the results the next time they are needed. The `addRefreshListener()` and `removeRefreshListener()` methods are not used by OpenOffice.org API Calc. The `refresh()` implementation in the example is empty, because the results are always calculated dynamically.

Dimensions

The `com.sun.star.sheet.DataPilotSourceDimensions` service contains an entry for each dimension that can be used as column, row or page dimension, for each possible data (measure) dimension, and one for the “data layout” dimension that contains the names of the data dimensions.

The example below initializes a dimension's orientation as DATA for the data dimension, and is otherwise HIDDEN. Thus, when the user creates a new DataPilot table using the example component, the data dimension is already present in the “Data” area of the DataPilot dialog. (Spreadsheet/ExampleDataPiloSource.java)

```
private ExampleSettings aSettings;
private int nDimension;
private com.sun.star.sheet.DataPilotFieldOrientation eOrientation;

public ExampleDimension(ExampleSettings aSet, int nDim) {
    aSettings = aSet;
    nDimension = nDim;
    eOrientation = (nDim == ExampleSettings.nValueDimension) ?
        com.sun.star.sheet.DataPilotFieldOrientation.DATA :
        com.sun.star.sheet.DataPilotFieldOrientation.HIDDEN;
}
```

The com.sun.star.sheet.DataPilotSourceDimension service contains a com.sun.star.beans.XPropertySet interface that is used for the following properties of a dimension:

- Original (read-only) contains the dimension object from which a dimension was cloned, or null if it was not cloned. A description of the com.sun.star.util.XCloneable interface is described below.
- IsDataLayoutDimension (read-only) must contain true if the dimension is the "data layout" dimension, otherwise false.
- Orientation controls how a dimension is used in the DataPilot table. If it contains the com.sun.star.sheet.DataPilotFieldOrientation enum values COLUMN or ROW, the dimension is used as a column or row dimension, respectively. If the value is DATA, the dimension is used as data (measure) dimension. The PAGE designates a page dimension, but is not currently used in OpenOffice.org API Calc. If the value is HIDDEN, the dimension is not used.
- Position contains the position of the dimension within the orientation. This controls the order of the dimensions. If a dimension's orientation is changed, it is added at the end of the dimensions for that orientation, and the Position property reflects that position.
- Function specifies the function that is used to aggregate data for a data dimension.
- UsedHierarchy selects which of the dimension's hierarchies is used in the DataPilot table. See the section on hierarchies below.
- Filter specifies a list of filter criteria to be applied to the source data before processing. It is currently not used by OpenOffice.org API Calc.

In the following example, the setPropertyValue() method for the dimension only implements the modification of Orientation and Position, using two lists to store the order of column and row dimensions. Page dimensions are not supported in the example.
(Spreadsheet/ExampleDataPiloSource.java)

```
public void setPropertyValue(String aPropertyName, Object aValue)
        throws com.sun.star.beans.UnknownPropertyException {
    if (aPropertyName.equals("Orientation")) {
        com.sun.star.sheet.DataPilotFieldOrientation eNewOrient =
            (com.sun.star.sheet.DataPilotFieldOrientation) aValue;
        if (nDimension != ExampleSettings.nValueDimension &&
                nDimension != ExampleSettings.nDataDimension &&
                eNewOrient != com.sun.star.sheet.DataPilotFieldOrientation.DATA) {

            // remove from list for old orientation and add for new one
            Integer aDimInt = new Integer(nDimension);
            if (eOrientation == com.sun.star.sheet.DataPilotFieldOrientation.COLUMN)
                aSettings.aColDimensions.remove(aDimInt);
            else if (eOrientation == com.sun.star.sheet.DataPilotFieldOrientation.ROW)
                aSettings.aRowDimensions.remove(aDimInt);
            if (eNewOrient == com.sun.star.sheet.DataPilotFieldOrientation.COLUMN)
                aSettings.aColDimensions.add(aDimInt);
            else if (eNewOrient == com.sun.star.sheet.DataPilotFieldOrientation.ROW)
                aSettings.aRowDimensions.add(aDimInt);

            // change orientation
            eOrientation = eNewOrient;
        }
    } else if (aPropertyName.equals("Position")) {
        int nNewPos = ((Integer) aValue).intValue();
        Integer aDimInt = new Integer(nDimension);
        if (eOrientation == com.sun.star.sheet.DataPilotFieldOrientation.COLUMN) {
            aSettings.aColDimensions.remove(aDimInt);
            aSettings.aColDimensions.add( nNewPos, aDimInt );
        }
        else if (eOrientation == com.sun.star.sheet.DataPilotFieldOrientation.ROW) {
            aSettings.aRowDimensions.remove(aDimInt);
            aSettings.aRowDimensions.add(nNewPos, aDimInt);
        }
    } else if (aPropertyName.equals("Function") || aPropertyName.equals("UsedHierarchy") ||
            aPropertyName.equals("Filter")) {
        // ignored
    } else
        throw new com.sun.star.beans.UnknownPropertyException();
}
```

The associated `getPropertyValue()` method returns the stored values for `Orientation` and `Position`. If it is the data layout dimension, then `IsDataLayoutDimension` is `true`, and the values default for the remaining properties. (Spreadsheet/ExampleDataPiloSource.java)

```
public Object getPropertyValue(String aPropertyName)
        throws com.sun.star.beans.UnknownPropertyException {
    if (aPropertyName.equals("Original"))
        return null;
    else if (aPropertyName.equals( "IsDataLayoutDimension"))
        return new Boolean(nDimension == ExampleSettings.nDataDimension);
    else if (aPropertyName.equals("Orientation"))
        return eOrientation;
    else if (aPropertyName.equals("Position")) {
        int nPosition;
        if (eOrientation == com.sun.star.sheet.DataPilotFieldOrientation.COLUMN)
            nPosition = aSettings.aColDimensions.indexOf(new Integer(nDimension));
        else if (eOrientation == com.sun.star.sheet.DataPilotFieldOrientation.ROW)
            nPosition = aSettings.aRowDimensions.indexOf(new Integer(nDimension));
        else
            nPosition = nDimension;
        return new Integer(nPosition);
    }
    else if (aPropertyName.equals("Function"))
        return com.sun.star.sheet.GeneralFunction.SUM;
    else if (aPropertyName.equals("UsedHierarchy"))
        return new Integer(0);
    else if (aPropertyName.equals("Filter"))
        return new com.sun.star.sheet.TableFilterField[0];
    else
        throw new com.sun.star.beans.UnknownPropertyException();
}
```

The dimension's `com.sun.star.util.XCloneable` interface is required when a dimension is used in multiple positions. The DataPilot dialog allows the use of a column or row dimension additionally as data dimension, and it also allows multiple use of a data dimension by assigning several functions to it. In both cases, additional dimension objects are created from the original one by calling the `createClone()` method. Each clone is given a new name using the `com.sun.star.container.XNamed` interface's `setName()` method, then the different settings are

applied to the objects. A dimension object that was created using the `createClone()` method must return the original object that it was created from in the `Original` property.

The example does not support multiple uses of a dimension, so it always returns `null` from the `createClone()` method, and the `Original` property is also always `null`.

Hierarchies

A single dimension can have several hierarchies, that is, several ways of grouping the elements of the dimension. For example, date values may be grouped:

- in a hierarchy with the levels “year”, “month” and “day of month”.
- in a hierarchy with the levels “year”, “week” and “day of week”.

The property `UsedHierarchy` of the `com.sun.star.sheet.DataPilotSourceDimension` service selects which hierarchy of a dimension is used. The property contains an index into the sequence of names that is returned by the dimension's `getElementNames()` method. OpenOffice.org API Calc currently has no user interface to select a hierarchy, so it uses the hierarchy that the initial value of the `UsedHierarchy` property selects.

The `com.sun.star.sheet.DataPilotSourceHierarchy` service serves as a container to access the levels object.

In the example, each dimension has only one hierarchy, which in turn has one level.

Levels

Each level of a hierarchy that is used in a DataPilot table corresponds to a column or row showing its members in the left or upper part of the table. The `com.sun.star.sheet.DataPilotSourceLevel` service contains a `com.sun.star.beans.XPropertySet` interface that is used to apply the following settings to a level:

- The `SubTotals` property defines a list of functions that are used to calculate subtotals for each member. If the sequence is empty, no subtotal columns or rows are generated. The `com.sun.star.sheet.GeneralFunction` enum value `AUTO` is used to select “automatic” subtotals, determined by the type of the data.
- The `ShowEmpty` property controls if result columns or rows are generated for members that have no data.

Both of these settings can be modified by the user in the “Data Field” dialog. The example does not use them.

The `com.sun.star.sheet.XDataPilotMemberResults` interface is used to get the result header column that is displayed below the level's name for a row dimension, or the header row for a column dimension. The sequence returned from the `getResults()` call must have the same size as the data result's columns or rows respectively, or be empty. If the sequence is empty, or none of the entries contains the `HASMEMBER` flag, the level is not shown.

The `com.sun.star.sheet.MemberResult` struct contains the following members:

- `Name` is the name of the member that is represented by the entry, exactly as returned by the member object's `getName()` method. It is used to find the member object, for example when the user double-clicks on the cell.
- `Caption` is the string that will be displayed in the cell. It may or may not be the same as `Name`.

- `Flags` indicates the kind of result the entry represents. It can be a combination of the `com.sun.star.sheet.MemberResultFlags` constants:
  - `HASMEMBER` indicates there is a member that belongs to this entry.
  - `SUBTOTAL` marks an entry that corresponds to a subtotal column or row. The `HASMEMBER` should be set.
  - `CONTINUE` marks an entry that is a continuation of the previous entry. In this case, none of the others are set, and the `Name` and `Caption` members are both empty.

In the example table shown above, the resulting sequence for the "ones" level would consist of:

- an entry containing the name and caption "1" and the `HASMEMBER` flag
- two entries containing only the `CONTINUE` flag
- the same repeated for member names "2" and "3".

The implementation for the example looks similar to this:
(Spreadsheet/ExampleDataPiloSource.java)

```
private ExampleSettings aSettings;
private int nDimension;

public com.sun.star.sheet.MemberResult[] getResults() {
    int nDimensions = 0;
    int nPosition = aSettings.aColDimensions.indexOf(new Integer(nDimension));
    if (nPosition >= 0)
        nDimensions = aSettings.aColDimensions.size();
    else {
        nPosition = aSettings.aRowDimensions.indexOf(new Integer(nDimension));
        if (nPosition >= 0)
            nDimensions = aSettings.aRowDimensions.size();
    }

    if (nPosition < 0)
        return new com.sun.star.sheet.MemberResult[0];

    int nMembers = aSettings.nMemberCount;
    int nRepeat = 1;
    int nFill = 1;
    for (int i=0; i<nDimensions; i++) {
        if (i < nPosition)
            nRepeat *= nMembers;
        else if (i > nPosition)
            nFill *= nMembers;
    }
    int nSize = nRepeat * nMembers * nFill;

    com.sun.star.sheet.MemberResult[] aResults = new com.sun.star.sheet.MemberResult[nSize];
    int nResultPos = 0;
    for (int nOuter=0; nOuter<nRepeat; nOuter++) {
        for (int nMember=0; nMember<nMembers; nMember++) {
            aResults[nResultPos] = new com.sun.star.sheet.MemberResult();
            aResults[nResultPos].Name = ExampleSettings.getMemberName( nMember );
            aResults[nResultPos].Caption = aResults[nResultPos].Name;
            aResults[nResultPos].Flags = com.sun.star.sheet.MemberResultFlags.HASMEMBER;
            ++nResultPos;

            for (int nInner=1; nInner<nFill; nInner++) {
                aResults[nResultPos] = new com.sun.star.sheet.MemberResult();
                aResults[nResultPos].Flags = com.sun.star.sheet.MemberResultFlags.CONTINUE;
                ++nResultPos;
            }
        }
    }
    return aResults;
}
```

Members

The `com.sun.star.sheet.DataPilotSourceMember` service contains two settings that are accessed through the `com.sun.star.beans.XPropertySet` interface:

- If the boolean `IsVisible` property is `false`, the member and its data are hidden. There is currently no user interface to change this property.
- The boolean `ShowDetails` property controls if the results for a member should be detailed in the following level. If a member has this property set to `false`, only a single result column or row is generated for each data dimension. The property can be changed by the user by double-clicking on a result header cell for the member.

These properties are not used in the example.

8.3.8 Protecting Spreadsheets

The interface `com.sun.star.document.XActionLockable` protects this cell from painting or updating during changes. The interface can be used to optimize the performance of complex changes, for instance, inserting or deleting formatted text.

The interface `com.sun.star.util.XProtectable` contains methods to protect and unprotect the spreadsheet with a password. Protecting the spreadsheet protects the locked cells only.

- The methods `protect()` and `unprotect()` to switch the protection on and off. If a wrong password is used to unprotect the spreadsheet, it leads to an exception.
- The method `isProtected()` returns the protection state of the spreadsheet as a `boolean` value.

8.3.9 Sheet Outline

The spreadsheet interface `com.sun.star.sheet.XSheetOutline` contains all the methods to control the row and column outlines of a spreadsheet:

Methods of `com.sun.star.sheet.XSheetOutline`	
`group()`	Creates a new outline group and the method `ungroup()` removes the innermost outline group for a cell range. The parameter `nOrientation` (type `com.sun.star.table.TableOrientation`) selects the orientation of the outline (columns or rows).
`autoOutline()`	Inserts outline groups for a cell range depending on formula references.
`clearOutline()`	Removes all outline groups from the sheet.
`hideDetail()`	Collapses an outline group.
`showDetail()`	Reopens an outline group.
`showLevel()`	Shows the specified number of outline group levels and hides the others.

8.3.10 Detective

The spreadsheet interface `com.sun.star.sheet.XSheetAuditing` supports the detective functionality of the spreadsheet.

Methods of `com.sun.star.sheet.XSheetAuditing`	
`hideDependents()` `hidePrecedents()`	Hides the last arrows to dependent or precedent cells of a formula cell. Repeated calls of the methods shrink the chains of arrows.

Methods of `com.sun.star.sheet.XSheetAuditing`	
`showDependents()` `showPrecedents()`	Adds arrows to the next dependent or precedent cells of a formula cell. Repeated calls of the methods extend the chains of arrows.
`showErrors()`	Inserts arrows to all cells that cause an error in the specified cell.
`showInvalid()`	Marks all cells that contain invalid values.
`clearArrows()`	Removes all auditing arrows from the spreadsheet.

8.3.11 Other Table Operations

Data Validation

Data validation checks if a user entered valid entries.

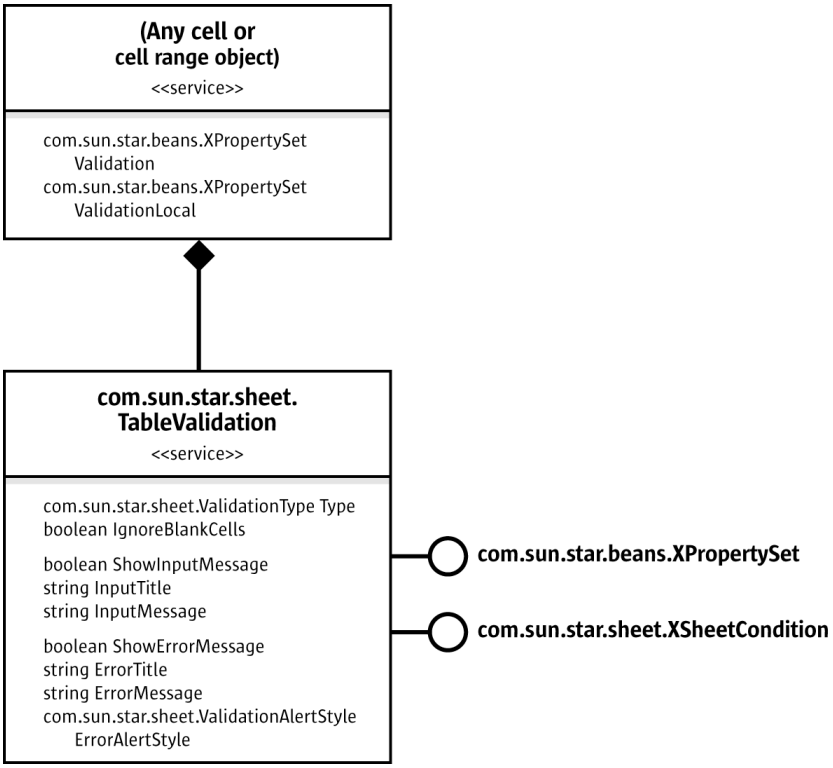


Illustration 115: TableValidation

A cell or cell range object contains the properties `Validation` and `ValidationLocal`. They return the interface `com.sun.star.beans.XPropertySet` of the validation object `com.sun.star.sheet.TableValidation`. The objects of both properties are equal, except the representation of formulas. The `ValidationLocal` property uses function names in the current language).



After the validation settings are changed, the validation object is reinserted into the property set of the cell or cell range.

- `Type` (type `com.sun.star.sheet.ValidationType`): Describes the type of data the cells contain. In text cells, it is possible to check the length of the text.

- `IgnoreBlankCells`: Determines if blank cells are valid.
- `ShowInputMessage`, `InputTitle` and `InputMessage`: These properties describe the message that appears if a cell of the validation area is selected.
- `ShowErrorMessage`, `ErrorTitle`, `ErrorMessage` and `ErrorAlertStyle` (type `com.sun.star.sheet.ValidationAlertStyle`): These properties describe the error message that appear if an invalid value has been entered. If the alert style is STOP, all invalid values are rejected. With the alerts WARNING and INFO, it is possible to keep invalid values. The alert MACRO starts a macro on invalid values. The property `ErrorTitle` has to contain the name of the macro.

The interface `com.sun.star.sheet.XSheetCondition` sets the conditions for valid values. The comparison operator, the first and second formula and the base address for relative references in formulas.

The following example enters values between 0.0 and 5.0 in a cell range. The `xSheet` is the interface `com.sun.star.sheet.XSpreadsheet` of a spreadsheet.
(Spreadsheet/SpreadsheetSample.java)

```
// --- Data validation ---
com.sun.star.table.XCellRange xCellRange = xSheet.getCellRangeByName("A7:C7");
com.sun.star.beans.XPropertySet xCellPropSet = (com.sun.star.beans.XPropertySet)
    UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, xCellRange);

// validation properties
com.sun.star.beans.XPropertySet xValidPropSet = (com.sun.star.beans.XPropertySet)
    xCellPropSet.getPropertyValue("Validation");
xValidPropSet.setPropertyValue("Type", com.sun.star.sheet.ValidationType.DECIMAL);
xValidPropSet.setPropertyValue("ShowErrorMessage", new Boolean(true));
xValidPropSet.setPropertyValue("ErrorMessage", "This is an invalid value!");
xValidPropSet.setPropertyValue("ErrorAlertStyle", com.sun.star.sheet.ValidationAlertStyle.STOP);

// condition
com.sun.star.sheet.XSheetCondition xCondition = (com.sun.star.sheet.XSheetCondition)
    UnoRuntime.queryInterface(com.sun.star.sheet.XSheetCondition.class, xValidPropSet);
xCondition.setOperator(com.sun.star.sheet.ConditionOperator.BETWEEN);
xCondition.setFormula1("0.0");
xCondition.setFormula2("5.0");

// apply on cell range
xCellPropSet.setPropertyValue("Validation", xValidPropSet);
```

Data Consolidation

The data consolidation feature calculates results based on several cell ranges.

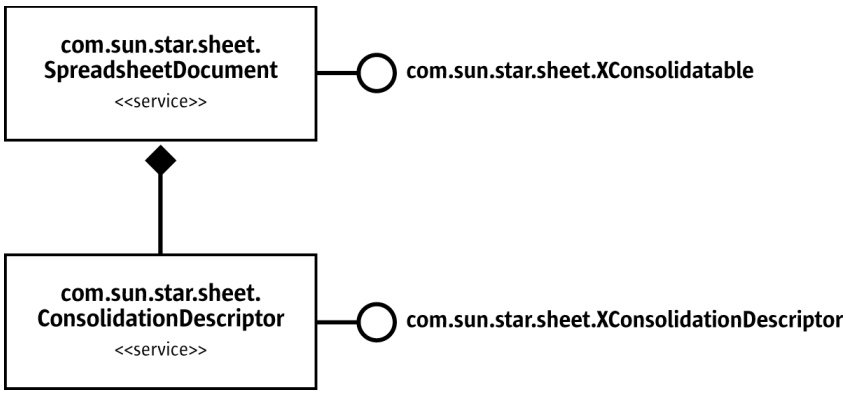


Illustration 116: ConsolidationDescriptor

The `com.sun.star.sheet.XConsolidatable`'s method `createConsolidationDescriptor()` returns the interface `com.sun.star.sheet.XConsolidationDescriptor` of a consolidation

descriptor (service `com.sun.star.sheet.ConsolidationDescriptor`). This descriptor contains all data needed for a consolidation. It is possible to get and set all properties:

- `getFunction()` and `setFunction()`: The function for calculation, type `com.sun.star.sheet.GeneralFunction`.
- `getSources()` and `setSources()`: A sequence of `com.sun.star.table.CellRangeAddress` structs with all cell ranges containing the source data.
- `getStartOutputPosition()` and `setStartOutputPosition()`: A `com.sun.star.table.CellAddress` containing the first cell of the result cell range.
- `getUseColumnHeaders()`, `setUseColumnHeaders()`, `getUseRowHeaders()` and `setUseRowHeaders()`: Determine if the first column or row of each cell range is used to find matching data.
- `getInsertLinks()` and `setInsertLinks()`: Determine if the results are linked to the source data (formulas are inserted) or not (only results are inserted).

The method `consolidate()` of the interface `com.sun.star.sheet.XConsolidatable` performs a consolidation with the passed descriptor.

Charts

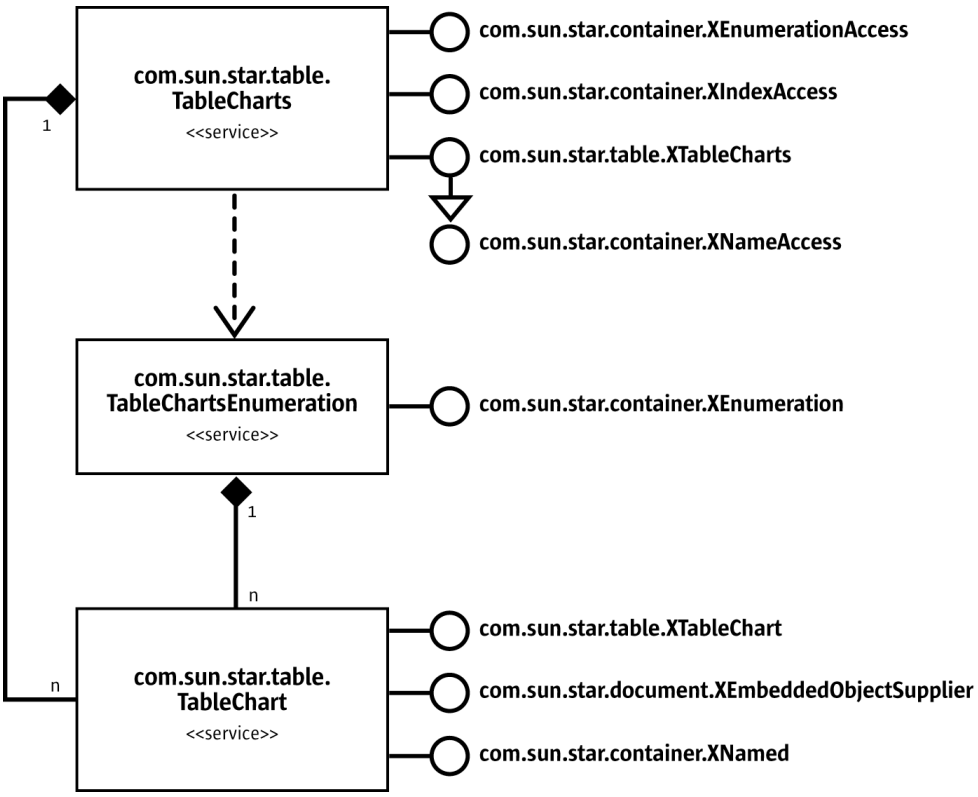


Illustration 117: TableCharts

The service `com.sun.star.table.TableChart` represents a chart object. The interface `com.sun.star.table.XTableChart` provides access to the cell range of the source data and controls the existence of column and row headers.



The service `com.sun.star.table.TableChart` does not represent the chart document, but the object in the table that contains the chart document. The interface `com.sun.star.document.XEmbeddedObjectSupplier` provides access to that chart document. For further information, see *10 Charts*.

The interface `com.sun.star.container.XNamed` retrieves and changes the name of the chart object.

For further information about charts, see *10 Charts*.

The service `com.sun.star.table.TableCharts` represents the collection of all chart objects contained in the table. It implements the interfaces:

- `com.sun.star.table.XTableCharts` to create new charts and accessing them by their names.
- `com.sun.star.container.XIndexAccess` to access the charts by the insertion index.
- `com.sun.star.container.XEnumerationAccess` to create an enumeration of all charts.

The following example shows how `xCharts` can be a `com.sun.star.table.XTableCharts` interface of a collection of charts. (Spreadsheet/GeneralTableSample.java)

```
// *** Inserting CHARTS ***
String aName = "newChart";
com.sun.star.awt.Rectangle aRect = new com.sun.star.awt.Rectangle();
aRect.X = 10000;
aRect.Y = 3000;
aRect.Width = aRect.Height = 5000;

com.sun.star.table.CellRangeAddress[] aRanges = new com.sun.star.table.CellRangeAddress[1];
aRanges[0] = new com.sun.star.table.CellRangeAddress();
aRanges[0].Sheet = aRanges[0].StartColumn = aRanges[0].EndColumn = 0;
aRanges[0].StartRow = 0; aRanges[0].EndRow = 9;

// Create the chart.
xCharts.addNewByName(aName, aRect, aRanges, false, false);

// Get the chart by name.
Object aChartObj = xCharts.getByName(aName);
com.sun.star.table.XTableChart xChart = (com.sun.star.table.XTableChart)
    UnoRuntime.queryInterface(com.sun.star.table.XTableChart.class, aChartObj);

// Query the state of row and column headers.
aText = "Chart has column headers: ";
aText += xChart.getHasColumnHeaders() ? "yes" : "no";
System.out.println(aText);
aText = "Chart has row headers: ";
aText += xChart.getHasRowHeaders() ? "yes" : "no";
System.out.println(aText);
```

Scenarios

A set of scenarios contains different selectable cell contents for one or more cell ranges in a spreadsheet. The data of each scenario in this set is stored in a hidden sheet following the scenario sheet. To change the scenario's data, its hidden sheet has to be modified.

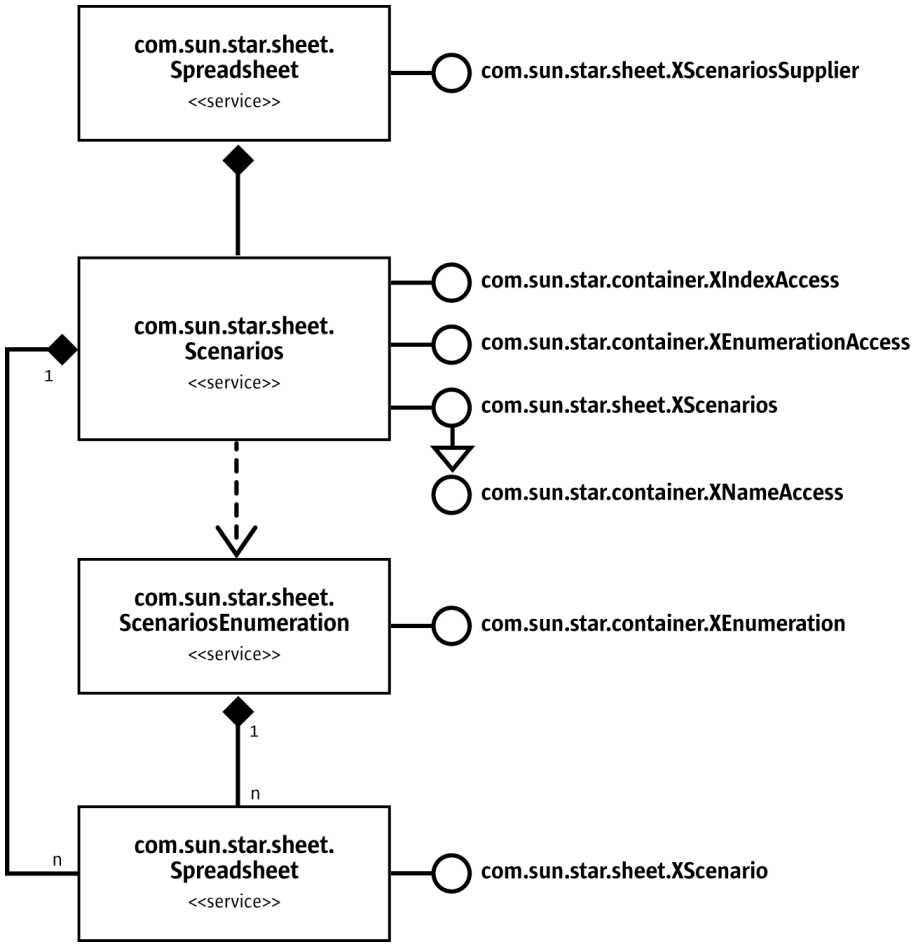


Illustration 118: Scenarios

The `com.sun.star.sheet.XScenariosSupplier`'s method `getScenarios()` returns the interface `com.sun.star.sheet.XScenarios` of the scenario set of the spreadsheet. This scenario set is represented by the service `com.sun.star.sheet.Scenarios` containing spreadsheet objects. It is possible to access the scenarios through their names that is equal to the name of the corresponding spreadsheet, their index, or using an enumeration (represented by the service `com.sun.star.sheet.ScenariosEnumeration`).

The interface `com.sun.star.sheet.XScenarios` inserts and removes scenarios:

- The method `addNewByName()` adds a scenario with the given name that contains the specified cell ranges.
- The method `removeByName()` removes the scenario (the spreadsheet) with the given name.

The following method shows how to create a scenario: (Spreadsheet/SpreadsheetSample.java)

```
/** Inserts a scenario containing one cell range into a sheet and applies the value array.
    @param xSheet           The XSpreadsheet interface of the spreadsheet.
    @param aRange           The range address for the scenario.
    @param aValueArray      The array of cell contents.
    @param aScenarioName    The name of the new scenario.
    @param aScenarioComment The user comment for the scenario.
 */
public void insertScenario(
        com.sun.star.sheet.XSpreadsheet xSheet,
        String aRange,
        Object[][] aValueArray,
        String aScenarioName,
        String aScenarioComment ) throws RuntimeException, Exception {
    // get the cell range with the given address
    com.sun.star.table.XCellRange xCellRange = xSheet.getCellRangeByName(aRange);

    // create the range address sequence
    com.sun.star.sheet.XCellRangeAddressable xAddr = (com.sun.star.sheet.XCellRangeAddressable)
        UnoRuntime.queryInterface(com.sun.star.sheet.XCellRangeAddressable.class, xCellRange);
    com.sun.star.table.CellRangeAddress[] aRangesSeq = new com.sun.star.table.CellRangeAddress[1];
    aRangesSeq[0] = xAddr.getRangeAddress();

    // create the scenario
    com.sun.star.sheet.XScenariosSupplier xScenSupp = (com.sun.star.sheet.XScenariosSupplier)
        UnoRuntime.queryInterface(com.sun.star.sheet.XScenariosSupplier.class, xSheet);
    com.sun.star.sheet.XScenarios xScenarios = xScenSupp.getScenarios();
    xScenarios.addNewByName(aScenarioName, aRangesSeq, aScenarioComment);

    // insert the values into the range
    com.sun.star.sheet.XCellRangeData xData = (com.sun.star.sheet.XCellRangeData)
        UnoRuntime.queryInterface(com.sun.star.sheet.XCellRangeData.class, xCellRange);
    xData.setDataArray(aValueArray);
}
```

The service `com.sun.star.sheet.Spreadsheet` implements the interface `com.sun.star.sheet.XScenario` to modify an existing scenario:

- The method `getIsScenario()` tests if this spreadsheet is used to store scenario data.
- The methods `getScenarioComment()` and `setScenarioComment()` retrieves and sets the user comment for this scenario.
- The method `addRanges()` adds new cell ranges to the scenario.
- The method `apply()` copies the data of this scenario to the spreadsheet containing the scenario set, that is, it makes the scenario visible.

The following method shows how to activate a scenario: (Spreadsheet/SpreadsheetSample.java)

```
/** Activates a scenario.
    @param xSheet           The XSpreadsheet interface of the spreadsheet.
    @param aScenarioName    The name of the scenario.
*/
public void showScenario( com.sun.star.sheet.XSpreadsheet xSheet,
        String aScenarioName) throws RuntimeException, Exception {
    // get the scenario set
    com.sun.star.sheet.XScenariosSupplier xScenSupp = (com.sun.star.sheet.XScenariosSupplier)
        UnoRuntime.queryInterface(com.sun.star.sheet.XScenariosSupplier.class, xSheet);
    com.sun.star.sheet.XScenarios xScenarios = xScenSupp.getScenarios();

    // get the scenario and activate it
    Object aScenarioObj = xScenarios.getByName(aScenarioName);
    com.sun.star.sheet.XScenario xScenario = (com.sun.star.sheet.XScenario)
        UnoRuntime.queryInterface(com.sun.star.sheet.XScenario.class, aScenarioObj);
    xScenario.apply();
}
```

8.4 Overall Document Features

8.4.1 Styles

A style contains all formatting properties for a specific object. All styles of the same type are contained in a collection named a *style family*. Each style family has a specific name to identify it in the collection. In OpenOffice.org API Calc, there are two style families named *CellStyles* and *PageStyles*. A cell style can be applied to a cell, a cell range, or all cells of the spreadsheet. A page style can be applied to a spreadsheet itself.

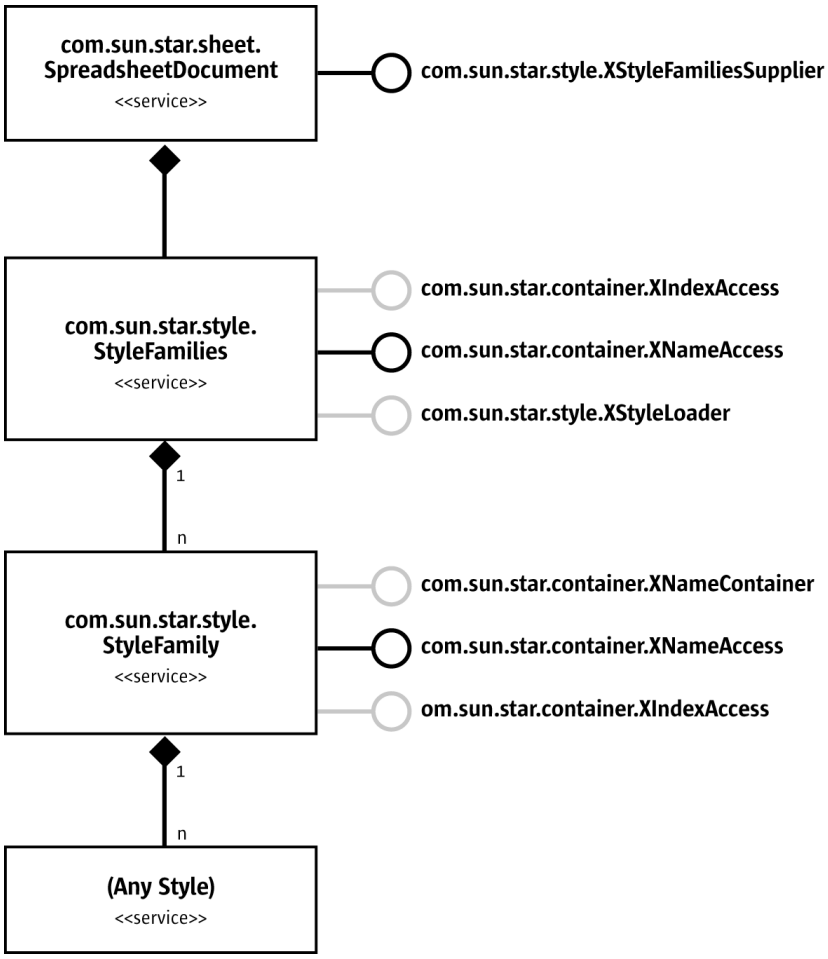


Illustration 119: StyleFamilies

The collection of style families is available from the spreadsheet document with the `com.sun.star.style.XStyleFamiliesSupplier`'s method `getStyleFamilies()`. The general handling of styles is described in *8.4.1 Spreadsheet Documents - Overall Document Features - Styles*, therefore this chapter focuses on the spreadsheet specific style properties.



A new style is inserted into the family container,then it is possible to set any properties.

Cell Styles

Cell styles are predefined packages of format settings that are applied in a single step.

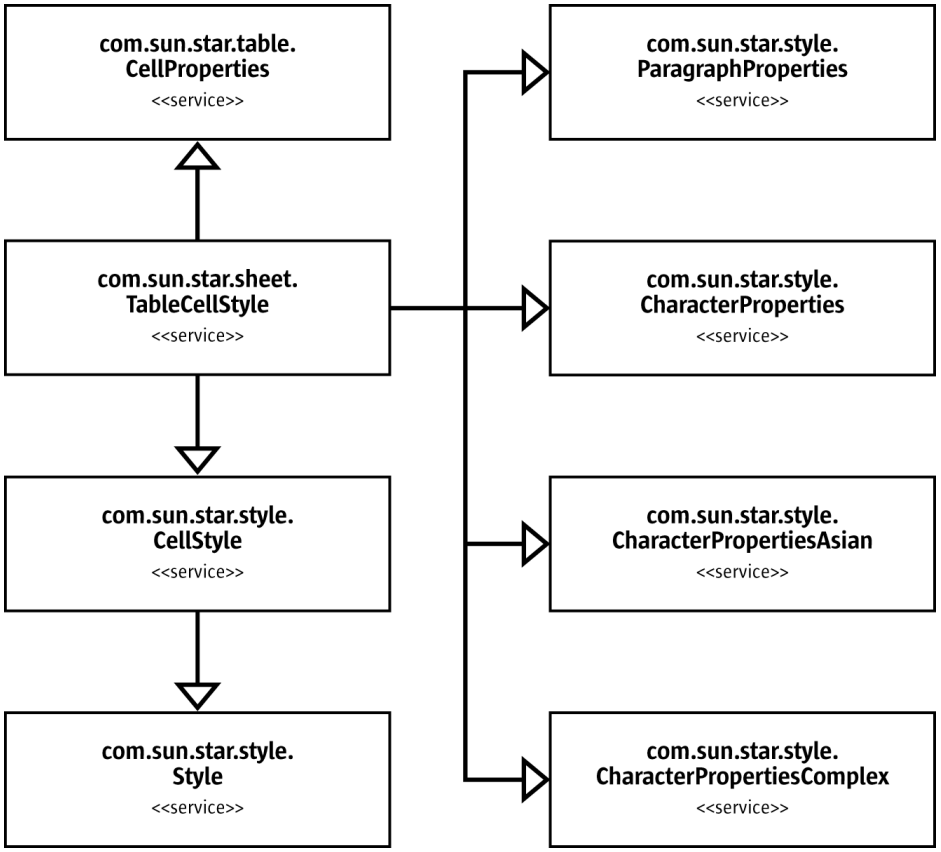


Illustration 120: CellStyle

A cell style is represented by the service `com.sun.star.sheet.TableCellStyle`. If a formatting property is applied directly to a cell, it covers the property of the applied cell style. This service does not support the property `CellStyle`. The name of the style is set with the interface `com.sun.star.container.XNamed`.

The following example creates a new cell style with gray background. The `xDocument` is the `com.sun.star.sheet.XSpreadsheetDocument` interface of a spreadsheet document. (Spreadsheet/SpreadsheetSample.java)

```
// get the cell style container
com.sun.star.style.XStyleFamiliesSupplier xFamiliesSupplier =
    (com.sun.star.style.XStyleFamiliesSupplier) UnoRuntime.queryInterface(
        com.sun.star.style.XStyleFamiliesSupplier.class, xDocument);
com.sun.star.container.XNameAccess xFamiliesNA = xFamiliesSupplier.getStyleFamilies();
Object aCellStylesObj = xFamiliesNA.getByName("CellStyles");
com.sun.star.container.XNameContainer xCellStylesNA = (com.sun.star.container.XNameContainer)
    UnoRuntime.queryInterface(com.sun.star.container.XNameContainer.class, aCellStylesObj);

// create a new cell style
com.sun.star.lang.XMultiServiceFactory xServiceManager = (com.sun.star.lang.XMultiServiceFactory)
    UnoRuntime.queryInterface(com.sun.star.lang.XMultiServiceFactory.class, xDocument);
Object aCellStyle = xServiceManager.createInstance("com.sun.star.style.CellStyle");
xCellStylesNA.insertByName("MyNewCellStyle", aCellStyle);

// modify properties of the new style
com.sun.star.beans.XPropertySet xPropSet = (com.sun.star.beans.XPropertySet)
    UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, aCellStyle);
xPropSet.setPropertyValue("CellBackColor", new Integer(0x888888));
xPropSet.setPropertyValue("IsCellBackgroundTransparent", new Boolean(false));
```

Page Styles

A page style is represented by the service `com.sun.star.sheet.TablePageStyle`. It contains the service `com.sun.star.style.PageStyle` and additional spreadsheet specific page properties.

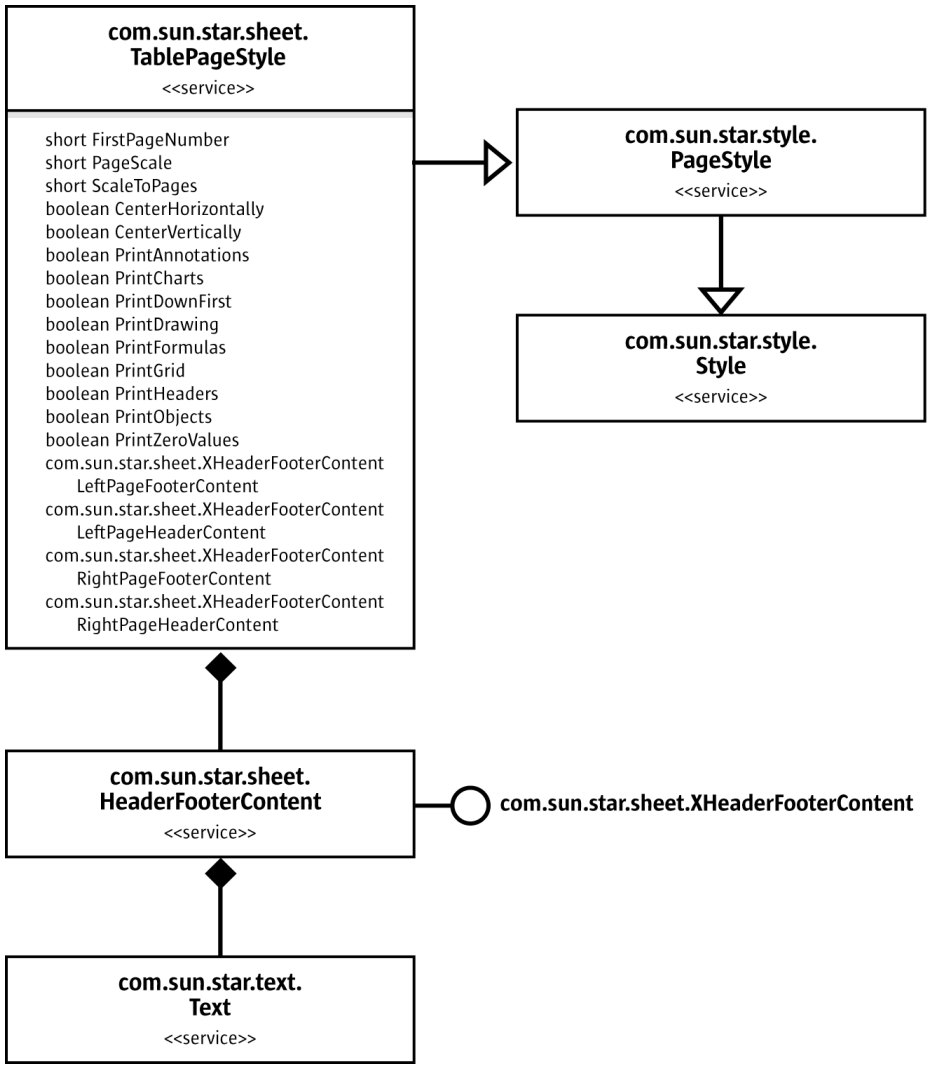


Illustration 121: TablePageStyle

The properties `LeftPageFooterContent`, `LeftPageHeaderContent`, `RightPageFooterContent` and `RightPageHeaderContent` return the interface `com.sun.star.sheet.XHeaderFooterContent` for the headers and footers for the left and right pages. Headers and footers are represented by the service `com.sun.star.sheet.HeaderFooterContent`. Each header or footer object contains three text objects for the left, middle and right portion of a header or footer. The methods `getLeftText()`, `getCenterText()` and `getRightText()` return the interface `com.sun.star.text.XText` of these text portions.



After the text of a header or footer is changed, it is reinserted into the property set of the page style.

8.4.2 Function Handling

This section describes the services which handle spreadsheet functions.

Calculating Function Results

The `com.sun.star.sheet.FunctionAccess` service calls any spreadsheet function and gets its result without having to insert a formula into a spreadsheet document.

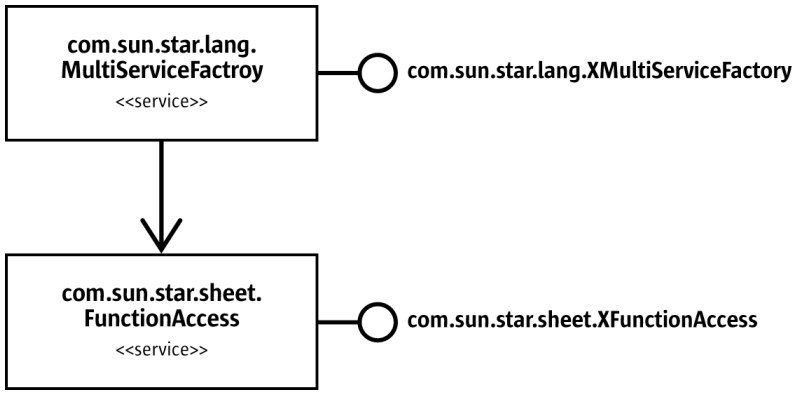


Illustration 122: FunctionAccess

The service can be instantiated through the service manager. The `com.sun.star.sheet.XFunctionAccess` interface contains only one method, `callFunction()`. The first parameter is the name of the function to call. The name has to be the function's programmatic name.

- For a built-in function, the English name is always used, regardless of the application's UI language.
- For an add-in function, the complete internal name that is the add-in component's service name, followed by a dot and the function's name as defined in the interface. For the `getIncremented` function in the example from the add-in section, this would be: "`com.sun.star.sheet.addin.ExampleAddIn.getIncremented`".

The second parameter to `callFunction()` is a sequence containing the function arguments. The supported types for each argument are described in the `com.sun.star.sheet.XFunctionAccess` interface description, and are similar to the argument types for add-in functions. The following example passes two arguments to the `ZTEST` function, an array of values and a single value. (Spreadsheet/SpreadsheetSample.java)

```
// --- Calculate a function ---
Object aFuncInst = xServiceManager.createInstance("com.sun.star.sheet.FunctionAccess");
com.sun.star.sheet.XFunctionAccess xFuncAcc = (com.sun.star.sheet.XFunctionAccess)
    UnoRuntime.queryInterface(com.sun.star.sheet.XFunctionAccess.class, aFuncInst);
// put the data into a two-dimensional array
double[][] aData = {{1.0, 2.0, 3.0}};
// construct the array of function arguments
Object[] aArgs = new Object[2];
aArgs[0] = aData;
aArgs[1] = new Double( 2.0 );
Object aResult = xFuncAcc.callFunction("ZTEST", aArgs);
System.out.println("ZTEST result for data {1,2,3} and value 2 is "
                                + ((Double)aResult).doubleValue());
```



The implementation of `com.sun.star.sheet.FunctionAccess` uses the same internal structures as a spreadsheet document, therefore it is bound by the same limitations, such as the limit of 32000 rows exist for the function arguments.

Information about Functions

The services `com.sun.star.sheet.FunctionDescriptions` and `com.sun.star.sheet.FunctionDescription` provide help texts about the available spreadsheet cell functions, including add-in functions and their arguments. This is the same information that OpenOffice.org API Calc displays in the function AutoPilot.

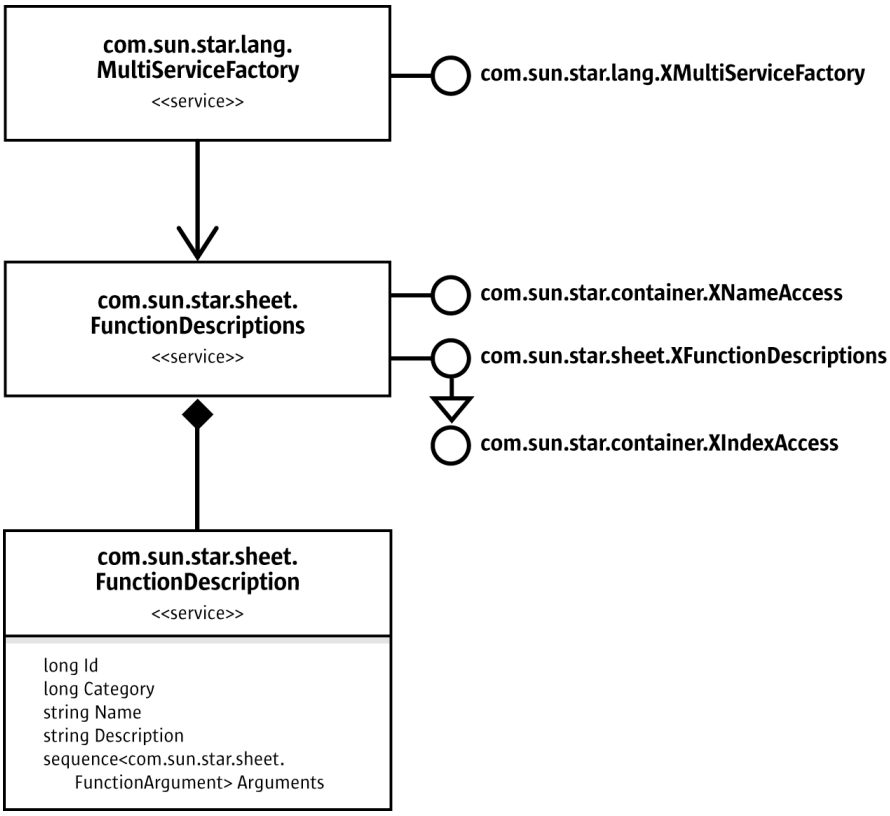


Illustration 123: FunctionDescriptions

The `com.sun.star.sheet.FunctionDescriptions` service is instantiated through the service manager. It provides three different methods to access the information for the different functions:

- By name through the `com.sun.star.container.XNameAccess` interface.
- By index through the `com.sun.star.container.XIndexAccess` interface.
- By function identifier through the `com.sun.star.sheet.XFunctionDescriptions` interface's `getById()` method. The function identifier is the same used in the `com.sun.star.sheet.RecentFunctions` service.

The `com.sun.star.sheet.FunctionDescription` that is returned by any of these calls is a sequence of `com.sun.star.beans.PropertyValue` structs. To access one of these properties, loop through the sequence, looking for the desired property's name in the `Name` member. The `Arguments` property contains a sequence of `com.sun.star.sheet.FunctionArgument` structs, one for each argument that the function accepts. The struct contains the name and description of the argument, as well as a boolean flag showing if the argument is optional.



All of the strings contained in the `com.sun.star.sheet.FunctionDescription` service are to be used in user interaction, and therefore translated to the application's UI language. They cannot be used where programmatic function names are expected, for example, the `com.sun.star.sheet.FunctionAccess` service.

The Recently Used Functions section below provides an example on how to use the `com.sun.star.sheet.FunctionDescriptions` service.

Recently Used Functions

The `com.sun.star.sheet.RecentFunctions` service provides access to the list of recently used functions of the spreadsheet application, that is displayed in the **AutoPilot:Functions** and the **Function List** window for example.

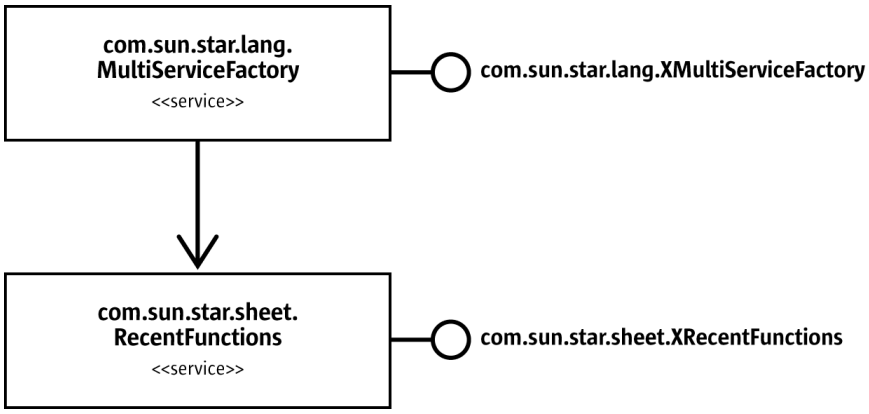


Illustration 124: RecentFunctions

The service can be instantiated through the service manager. The `com.sun.star.sheet.XRecentFunctions` interface's `getRecentFunctionIds()` method returns a sequence of function identifiers that are used with the `com.sun.star.sheet.FunctionDescriptions` service. The `setRecentFunctionIds()` method changes the list. If the parameter to the `setRecentFunctionIds()` call contains more entries than the application handles, only the first entries are used. The maximum size of the list of recently used functions, currently 10, can be queried with the `getMaxRecentFunctions()` method.

The following example demonstrates the use of the `com.sun.star.sheet.RecentFunctions` and `com.sun.star.sheet.FunctionDescriptions` services. (Spreadsheet/SpreadsheetSample.java)

```
// --- Get the list of recently used functions ---
Object aRecInst = xServiceManager.createInstance("com.sun.star.sheet.RecentFunctions");
com.sun.star.sheet.XRecentFunctions xRecFunc = (com.sun.star.sheet.XRecentFunctions)
    UnoRuntime.queryInterface(com.sun.star.sheet.XRecentFunctions.class, aRecInst);
int[] nRecentIds = xRecFunc.getRecentFunctionIds();

// --- Get the names for these functions ---
Object aDescInst = xServiceManager.createInstance("com.sun.star.sheet.FunctionDescriptions");
com.sun.star.sheet.XFunctionDescriptions xFuncDesc = (com.sun.star.sheet.XFunctionDescriptions)
    UnoRuntime.queryInterface(com.sun.star.sheet.XFunctionDescriptions.class, aDescInst);
System.out.print("Recently used functions: ");
for (int nFunction=0; nFunction<nRecentIds.length; nFunction++) {
    com.sun.star.beans.PropertyValue[] aProperties = xFuncDesc.getById(nRecentIds[nFunction]);
    for (int nProp=0; nProp<aProperties.length; nProp++)
        if (aProperties[nProp].Name.equals("Name"))
            System.out.print(aProperties[nProp].Value + " ");
}
System.out.println();
```

8.4.3 Settings

The `com.sun.star.sheet.GlobalSheetSettings` service contains settings that affect the whole spreadsheet application. It can be instantiated through the service manager. The properties are accessed using the `com.sun.star.beans.XPropertySet` interface.

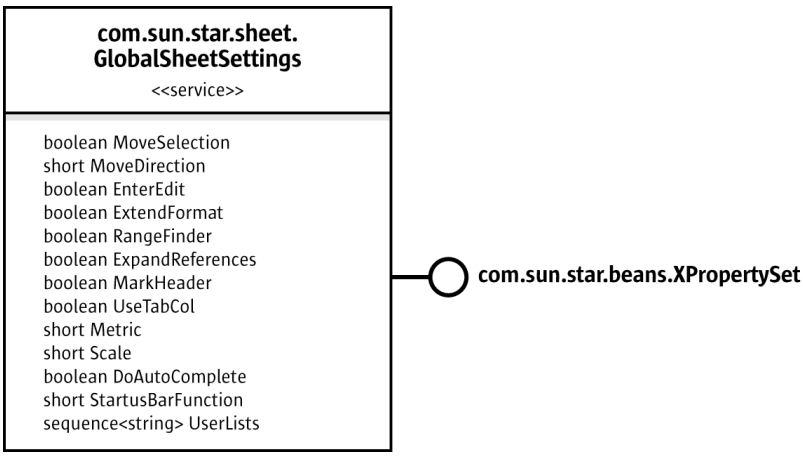


Illustration 125: GlobalSheetSettings

The following example gets the list of user-defined sort lists from the settings and displays them: (Spreadsheet/SpreadsheetSample.java)

```
// --- Get the user defined sort lists ---
 Object aSettings = xServiceManager.createInstance("com.sun.star.sheet.GlobalSheetSettings");
 com.sun.star.beans.XPropertySet xPropSet = (com.sun.star.beans.XPropertySet)
     UnoRuntime.queryInterface( com.sun.star.beans.XPropertySet.class, aSettings );
 String[] aEntries = (String[]) xPropSet.getPropertyValue("UserLists");
 System.out.println("User defined sort lists:");
 for (int i=0; i<aEntries.length; i++)
     System.out.println( aEntries[i] );
```

8.5 Spreadsheet Document Controller

8.5.1 Spreadsheet View

The `com.sun.star.sheet.SpreadsheetView` service is the spreadsheet's extension of the `com.sun.star.frame.Controller` service and represents a table editing view for a spreadsheet document.

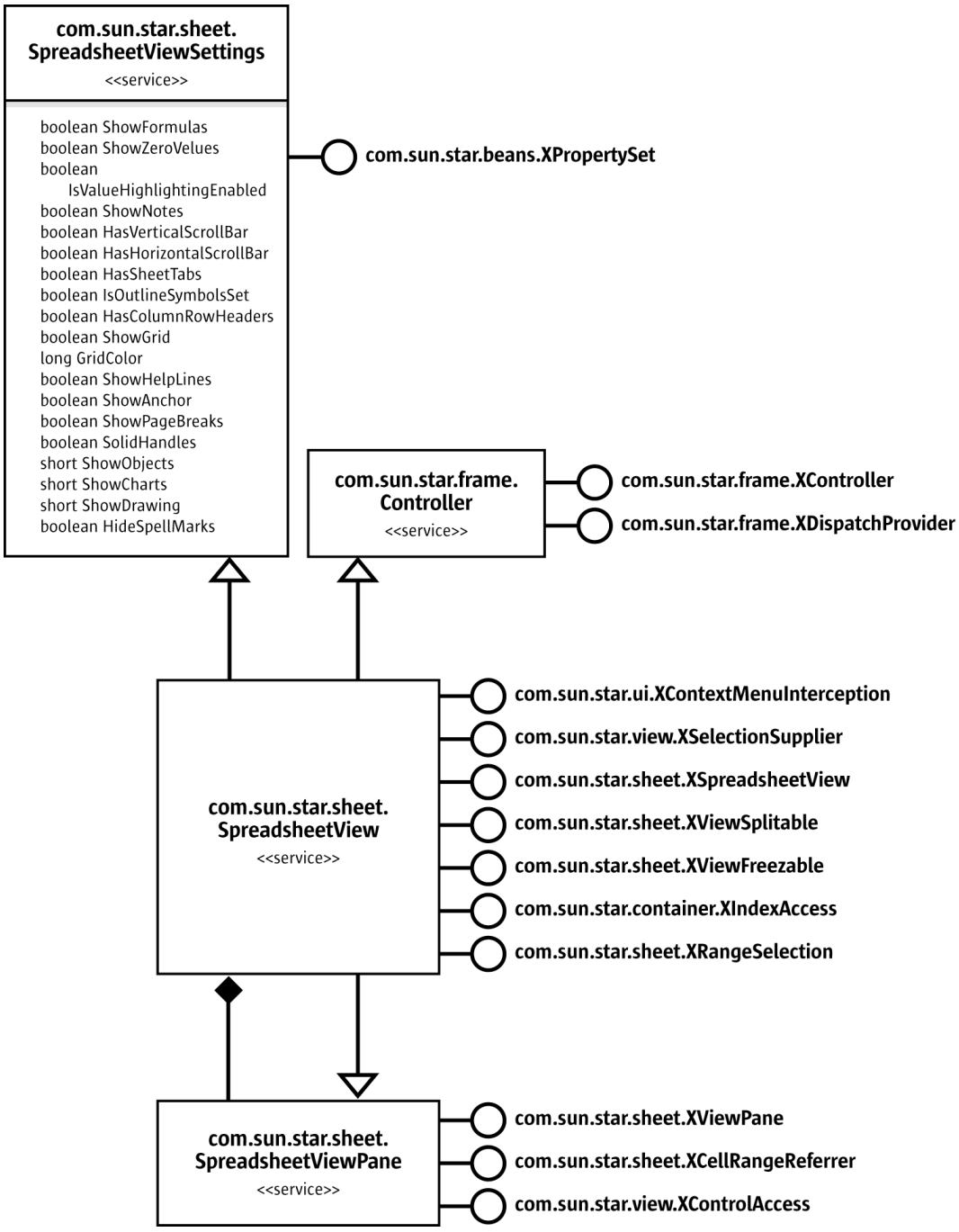


Illustration 126: SpreadsheetView



The page preview does not have an API representation.

The view object is the spreadsheet application's controller object as described in the chapter *6.1.1 Office Development - OpenOffice.org Application Environment - Overview - Framework API - Frame-Controller-Model Paradigm*. The `com.sun.star.frame.XController`, `com.sun.star.frame.XDispatchProvider` and `com.sun.star.ui.XContextMenuInterception` interfaces work as described in that chapter.

The com.sun.star.view.XSelectionSupplier interface queries and modifies the view's selection. The selection in a spreadsheet view can be a com.sun.star.sheet.SheetCell, com.sun.star.sheet.SheetCellRange, com.sun.star.sheet.SheetCellRanges, com.sun.star.drawing.Shape or com.sun.star.drawing.Shapes object.

The com.sun.star.sheet.XSpreadsheetView interface gives access to the spreadsheet that is displayed in the view. The getActiveSheet() method returns the active sheet's object, the setActiveSheet() method switches to a different sheet. The parameter to setActiveSheet() must be a sheet of the view's document.

The com.sun.star.sheet.XViewSplitable interface splits a view into two parts or panes, horizontally and vertically. The splitAtPosition() method splits the view at the specified pixel positions. To remove the split, a position of 0 is passed. The getIsWindowSplit() method returns true if the view is split, the getSplitHorizontal() and getSplitVertical() methods return the pixel positions where the view is split. The getSplitColumn() and getSplitRow() methods return the cell column or row that corresponds to the split position, and are used with frozen panes as discussed below.

The com.sun.star.sheet.XViewFreezable interface is used to freeze a number of columns and rows in the left and upper part of the view. The freezeAtPosition() method freezes the specified number of columns and rows. This also sets the split positions accordingly. The hasFrozenPanes() method returns true if the columns or rows are frozen. A view can only have frozen columns or rows, or normal split panes at a time.

If a view is split or frozen, it has up to four view pane objects that represent the individual parts. These are accessed using the com.sun.star.container.XIndexAccess interface. If a view is not split, it contains only one pane object. The active pane of a spreadsheet view is also accessed using the com.sun.star.sheet.SpreadsheetViewPane service's interfaces directly with the com.sun.star.sheet.SpreadsheetView service that inherits them.

The com.sun.star.sheet.XRangeSelection interface is explained in the “Range Selection” chapter below.

The following example uses the com.sun.star.sheet.XViewFreezable interface to freeze the first column and the first two rows: (Spreadsheet/ViewSample.java)

```
// freeze the first column and first two rows
com.sun.star.sheet.XViewFreezable xFreeze = (com.sun.star.sheet.XViewFreezable)
    UnoRuntime.queryInterface(com.sun.star.sheet.XViewFreezable.class, xController);
xFreeze.freezeAtPosition(1, 2);
```

8.5.2 View Panes

The com.sun.star.sheet.SpreadsheetViewPane service represents a pane in a view that shows a rectangular area of the document. The exposed area of a view pane always starts at a cell boundary. The com.sun.star.sheet.XViewPane interface's getFirstVisibleColumn(), getFirstVisibleRow(), setFirstVisibleColumn() and setFirstVisibleRow() methods query and set the start of the exposed area. The getVisibleRange() method returns a com.sun.star.table.CellRangeAddress struct describing which cells are shown in the pane. Columns or rows that are only partly visible at the right or lower edge of the view are not included.

The com.sun.star.sheet.XCellRangeReferrer interface gives direct access to the same cell range of exposed cells that are addressed by the getVisibleRange() return value.

The com.sun.star.view.XControlAccess interface's getControl() method gives access to a control model's control for the view pane. Refer to the chapter *13.2 Forms - Models and Views* for additional information.

The example below retrieves the cell range that is shown in the second pane. It is the lower left one after freezing both columns and rows, and assigns a cell background:
(Spreadsheet/ViewSample.java)

```
// get the cell range shown in the second pane and assign a cell background to them
com.sun.star.container.XIndexAccess xIndex = (com.sun.star.container.XIndexAccess)
    UnoRuntime.queryInterface(com.sun.star.container.XIndexAccess.class, xController);
Object aPane = xIndex.getByIndex(1);
com.sun.star.sheet.XCellRangeReferrer xRefer = (com.sun.star.sheet.XCellRangeReferrer)
    UnoRuntime.queryInterface(com.sun.star.sheet.XCellRangeReferrer.class, aPane);
com.sun.star.table.XCellRange xRange = xRefer.getReferredCells();
com.sun.star.beans.XPropertySet xRangeProp = (com.sun.star.beans.XPropertySet)
    UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, xRange);
xRangeProp.setPropertyValue("IsCellBackgroundTransparent", new Boolean(false));
xRangeProp.setPropertyValue("CellBackColor", new Integer(0xFFFFCC));
```

8.5.3 View Settings

The properties from the `com.sun.star.sheet.SpreadsheetViewSettings` service are accessed through the `com.sun.star.beans.XPropertySet` interface controlling the appearance of the view. Most of the properties correspond to settings in the options dialog. The `ShowObjects`, `ShowCharts` and `ShowDrawing` properties take values of 0 for “show”, 1 for “hide”, and 2 for “placeholder display”.

The following example changes the view to display green grid lines:
(Spreadsheet/ViewSample.java)

```
// change the view to display green grid lines
com.sun.star.beans.XPropertySet xProp = (com.sun.star.beans.XPropertySet)
    UnoRuntime.queryInterface(com.sun.star.beans.XPropertySet.class, xController);
xProp.setPropertyValue("ShowGrid", new Boolean(true));
xProp.setPropertyValue("GridColor", new Integer(0x00CC00));
```

8.5.4 Range Selection

The view's `com.sun.star.sheet.XRangeSelection` interface is used to let a user interactively select a cell range in the view, independently of the view's selection. This is used for dialogs that require a cell reference as input. While the range selection is active, a small dialog is shown, similar to the minimized state of OpenOffice.org API's own dialogs that allow cell reference input.

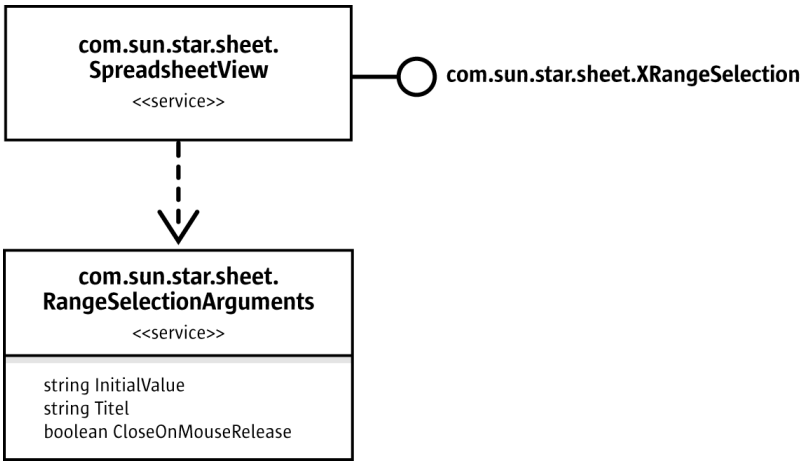


Illustration 127: XRangeSelection interface

Before the range selection mode is started, a listener is registered using the `addRangeSelectionListener()` method. The listener implements the `com.sun.star.sheet.XRangeSelectionListener` interface. Its `done()` or `aborted()` method is called when the selection is finished or aborted. The `com.sun.star.sheet.RangeSelectionEvent` struct that is passed to the calls contains the selected range in the `RangeDescriptor` member. It is a string because the user can type into the minimized dialog during range selection.

In the following example, the listener implementation stores the result in a member in the `done()` method, and notifies the main thread about the completion of the selection in the `done()` and `aborted()` methods: (Spreadsheet/ViewSample.java)

```
private class ExampleRangeListener implements com.sun.star.sheet.XRangeSelectionListener {
    public String aResult;

    public void done(com.sun.star.sheet.RangeSelectionEvent aEvent) {
        aResult = aEvent.RangeDescriptor;
        synchronized (this) {
            notify();
        }
    }

    public void aborted( com.sun.star.sheet.RangeSelectionEvent aEvent ) {
        synchronized (this) {
            notify();
        }
    }

    public void disposing( com.sun.star.lang.EventObject aObj ) {
    }
}
```

It is also possible to add another listener using the `addRangeSelectionChangeListener()` method. This listener implements the `com.sun.star.sheet.XRangeSelectionChangeListener` interface, and its `descriptorChanged()` method is called during the selection when the selection changes. Using this listener normally is not necessary.

After registering the listeners, the range selection mode is started using the `startRangeSelection()` method. The parameter to that method is a sequence of property values with properties from the `com.sun.star.sheet.RangeSelectionArguments` service:

- `InitialValue` specifies an existing selection value that is shown in the dialog and highlighted in the view when the selection mode is started.
- `Title` is the title for the range selection dialog.
- `CloseOnMouseRelease` specifies when the selection mode is ended. If the value is `true`, selection is ended when the mouse button is released after selecting a cell range. If it is `false` or not specified, the user presses the **Shrink** button in the dialog to end selection mode.

The `startRangeSelection()` method returns immediately after starting the range selection mode. This allows it to be called from a dialog's event handler. The `abortRangeSelection()` method is used to cancel the range selection mode programmatically.

The following example lets the user pick a range, and then selects that range in the view. Note that the use of `wait` to wait for the end of the selection is not how a GUI application normally handles the events. (Spreadsheet/ViewSample.java)

```
    // let the user select a range and use it as the view's selection
    com.sun.star.sheet.XRangeSelection xRngSel = (com.sun.star.sheet.XRangeSelection)
        UnoRuntime.queryInterface(com.sun.star.sheet.XRangeSelection.class, xController);
    ExampleRangeListener aListener = new ExampleRangeListener();
    xRngSel.addRangeSelectionListener(aListener);
    com.sun.star.beans.PropertyValue[] aArguments = new com.sun.star.beans.PropertyValue[2];
    aArguments[0] = new com.sun.star.beans.PropertyValue();
    aArguments[0].Name   = "Title";
    aArguments[0].Value  = "Please select a range";
    aArguments[1] = new com.sun.star.beans.PropertyValue();
    aArguments[1].Name   = "CloseOnMouseRelease";
    aArguments[1].Value  = new Boolean(true);
    xRngSel.startRangeSelection(aArguments);
    synchronized (aListener) {
        aListener.wait();       // wait until the selection is done
    }
    xRngSel.removeRangeSelectionListener(aListener);
    if (aListener.aResult != null && aListener.aResult.length() != 0)
    {
        com.sun.star.view.XSelectionSupplier xSel = (com.sun.star.view.XSelectionSupplier)
            UnoRuntime.queryInterface(com.sun.star.view.XSelectionSupplier.class, xController);
        com.sun.star.sheet.XSpreadsheetView xView = (com.sun.star.sheet.XSpreadsheetView)
            UnoRuntime.queryInterface(com.sun.star.sheet.XSpreadsheetView.class, xController);
        com.sun.star.sheet.XSpreadsheet xSheet = xView.getActiveSheet();
        com.sun.star.table.XCellRange xResultRange = xSheet.getCellRangeByName(aListener.aResult);
        xSel.select(xResultRange);
    }
```

8.6 Spreadsheet Add-Ins

An add-in component is used to add new functions to the spreadsheet application that can be used in cell formulas, such as the built-in functions. A spreadsheet add-in is a UNO component. The chapter *4 Writing UNO Components* describes how to write and deploy a UNO component.

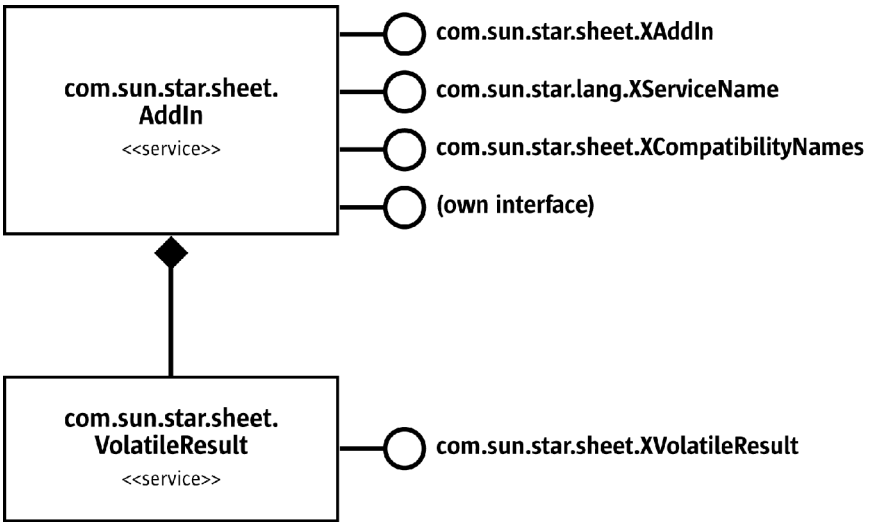


Illustration 128: AddIn

The functions that the add-in component exports to the spreadsheet application have to be defined in a new interface. The function names in the interface, together with the component's service name, are used internally to identify an add-in function. For a list of the supported types for function arguments and return values, see the `com.sun.star.sheet.AddIn` service description. An example interface that defines two functions is similar to the following code: (Spreadsheet/XExampleAddIn.idl)

```
#include <com/sun/star/uno/XInterface.idl>
#include <com/sun/star/sheet/XVolatileResult.idl>

module com { module sun { module star { module sheet { module addin {

    interface XExampleAddIn : com::sun::star::uno::XInterface
    {
        /// Sample function that just increments a value.
        long getIncremented( [in] long nValue );

        /// Sample function that returns a volatile result.
        com::sun::star::sheet::XVolatileResult getCounter( [in] string aName );
    };

}; }; }; }; };
```

In addition to this interface, the add-in has to implement the interfaces from the `com.sun.star.sheet.AddIn` service and the usual interfaces every component has to support.

8.6.1 Function Descriptions

The methods from the `com.sun.star.sheet.XAddIn` interface are used to provide descriptions of the user-visible functions.

The `getDisplayFunctionName()` and `getProgrammaticFuntionName()` methods are used to map between the internal function name, as defined in the interface and the function name as shown to the user of the spreadsheet application. The user-visible name, as well as the function and argument descriptions, can be translated strings for the language which is set using `setLocale()`.

The `getProgrammaticCategoryName()` method sorts each add-in functions into one of the spreadsheet application's function categories. It returns the category's internal (non-translated) name. In addition, the `getDisplayCategoryName()` method provides a translated name for the category.

The `getFunctionDescription()`, `getDisplayArgumentName()` and `getArgumentDescription()` methods provide descriptions of the function and its arguments that are shown to the user, for example in the function AutoPilot.



The `getProgrammaticFuntionName()` method name is misspelled, but the wrong spelling has to be retained for compatibility reasons.

8.6.2 Service Names

The add-in component has to support two services, the `com.sun.star.sheet.AddIn` service, and an additional service that is used to identify the set of functions that the add-in supplies. There may be several implementations of the same set of functions. In that case, they all use the same service name, but different implementation names. Therefore, a spreadsheet document that uses the functions can make use of the implementation that is present.

The `com.sun.star.lang.XServiceInfo` methods `supportsService()` and `getSupportedServiceNames()` handle both service names, and the component also has to be registered for both services. In addition, the component has to implement the `com.sun.star.lang.XServiceName` interface, and in its `getServiceName()` method return the name of the function-specific service.

8.6.3 Compatibility Names

Optionally, the component can implement the `com.sun.star.sheet.XCompatibilityNames` interface, and in the `getCompatibilityNames()` method return a sequence of locale-dependent compatibility names for a function. These names are used by the spreadsheet application when loading or saving Excel files. They should only be present for a function if it is known to be an Excel add-in function with equivalent functionality.

The sequence of compatibility names for a function may contain several names for a single locale. In that case, all of these names are considered when importing a file. When exporting, the first name is used. If a file is exported in a locale for which no entry is present, the first entry is used. If there is a default locale, the entries for that locale are first in the sequence.

8.6.4 Custom Functions

The user-visible functions have to be implemented as defined in the interface. The spreadsheet application does the necessary conversions to pass the arguments. For example, floating point numbers are rounded if a function has integer arguments. To enable the application to find the functions, it is important that the component implements the `com.sun.star.lang.XTypeProvider` interface.

The `getIncremented()` function from the example interface above can be implemented like this: (Spreadsheet/ExampleAddIn.java)

```
public int getIncremented( int nValue ) {
    return nValue + 1;
}
```

8.6.5 Variable Results

It is also possible to implement functions with results that change over time. Whenever such a result changes, the formulas that use the result are recalculated and the new values are shown in the spreadsheet. This can be used to display data from a real-time data feed in a spreadsheet.

In its interface, a function with a variable result must be defined with a return type of `com.sun.star.sheet.XVolatileResult`, such as the `getCounter()` function from the example interface above. The function's implementation must return an object that implements the `com.sun.star.sheet.VolatileResult` service. Subsequent calls to the same function with the same arguments return the same object. An implementation that returns a different result object for every name looks like this: (Spreadsheet/ExampleAddIn.java)

```
private java.util.Hashtable aResults = new java.util.Hashtable();

public com.sun.star.sheet.XVolatileResult getCounter(String aName) {
    ExampleAddInResult aResult = (ExampleAddInResult) aResults.get(aName);
    if (aResult == null) {
        aResult = new ExampleAddInResult(aName);
        aResults.put(aName, aResult);
    }
    return aResult;
}
```

The result object has to implement the `addResultListener()` and `removeResultListener()` methods from the `com.sun.star.sheet.XVolatileResult` interface to maintain a list of listeners, and notify each of these listeners by calling the `com.sun.star.sheet.XResultListener` interface's `modified()` method whenever a new result is available. The `com.sun.star.sheet.ResultEvent` object that is passed to the `modified()` call must contain the new result in the `Value` member. The possible types for the result are the same as for a function's return value if no volatile results are involved.

If a result is already available when `addResultListener()` is called, it can be publicized by immediately calling `modified()` for the new listener. Otherwise, the spreadsheet application displays a "#N/A" error value until a result is available.

The following example shows a simple implementation of a result object. Every time the incrementValue method is called, for example, from a background thread, the result value is incremented and the listeners are notified. (Spreadsheet/ExampleAddIn.java)

```
class ExampleAddInResult implements com.sun.star.sheet.XVolatileResult {
    private String aName;
    private int nValue;
    private java.util.Vector aListeners = new java.util.Vector();

    public ExampleAddInResult(String aNewName) {
        aName = aNewName;
    }

    private com.sun.star.sheet.ResultEvent getResult() {
        com.sun.star.sheet.ResultEvent aEvent = new com.sun.star.sheet.ResultEvent();
        aEvent.Value = aName + " " + String.valueOf(nValue);
        aEvent.Source = this;
        return aEvent;
    }

    public void addResultListener(com.sun.star.sheet.XResultListener aListener) {
        aListeners.addElement(aListener);

        // immediately notify of initial value
        aListener.modified(getResult());
    }

    public void removeResultListener(com.sun.star.sheet.XResultListener aListener) {
        aListeners.removeElement(aListener);
    }

    public void incrementValue() {
        ++nValue;
        com.sun.star.sheet.ResultEvent aEvent = getResult();

        java.util.Enumeration aEnum = aListeners.elements();
        while (aEnum.hasMoreElements())
            ((com.sun.star.sheet.XResultListener)aEnum.nextElement()).modified(aEvent);
    }
}
```

9 Drawing Documents and Presentation Documents

9.1 Overview

Draw and Impress are vector-oriented applications with the ability to create drawings and presentations. The drawing capabilities of Draw and Impress are identical. Both programs support a number of different shape types, such as rectangle, text, curve, or graphic shapes, that can be edited and arranged in various ways. Impress offers a presentation functionality where Draw does not. Impress is the ideal application to create and show presentations. It supports special presentation features, such as an enhanced page structure, presentation objects, and many slide transition and object effects. Draw is especially adapted for printed or standalone graphics, whereas Impress is optimized to fit screen dimensions and offers effects for business presentations.

The following diagrams show the document structure of Draw and Impress Documents.

In contrast to text documents and spreadsheet documents, the main content of drawing and presentation documents are their draw pages. Therefore the illustrations show the draw page container as integral part of the drawing and presentation document model. The drawing elements on the draw pages have to be created by the document service manager and are added to the draw pages afterwards.

Note the master pages and the layer manager, which are specific to drawings and presentations. Like for texts and spreadsheets, a controller is used to present the drawing in the GUI and services for styles and layout are available to handle overall document features such as styles.

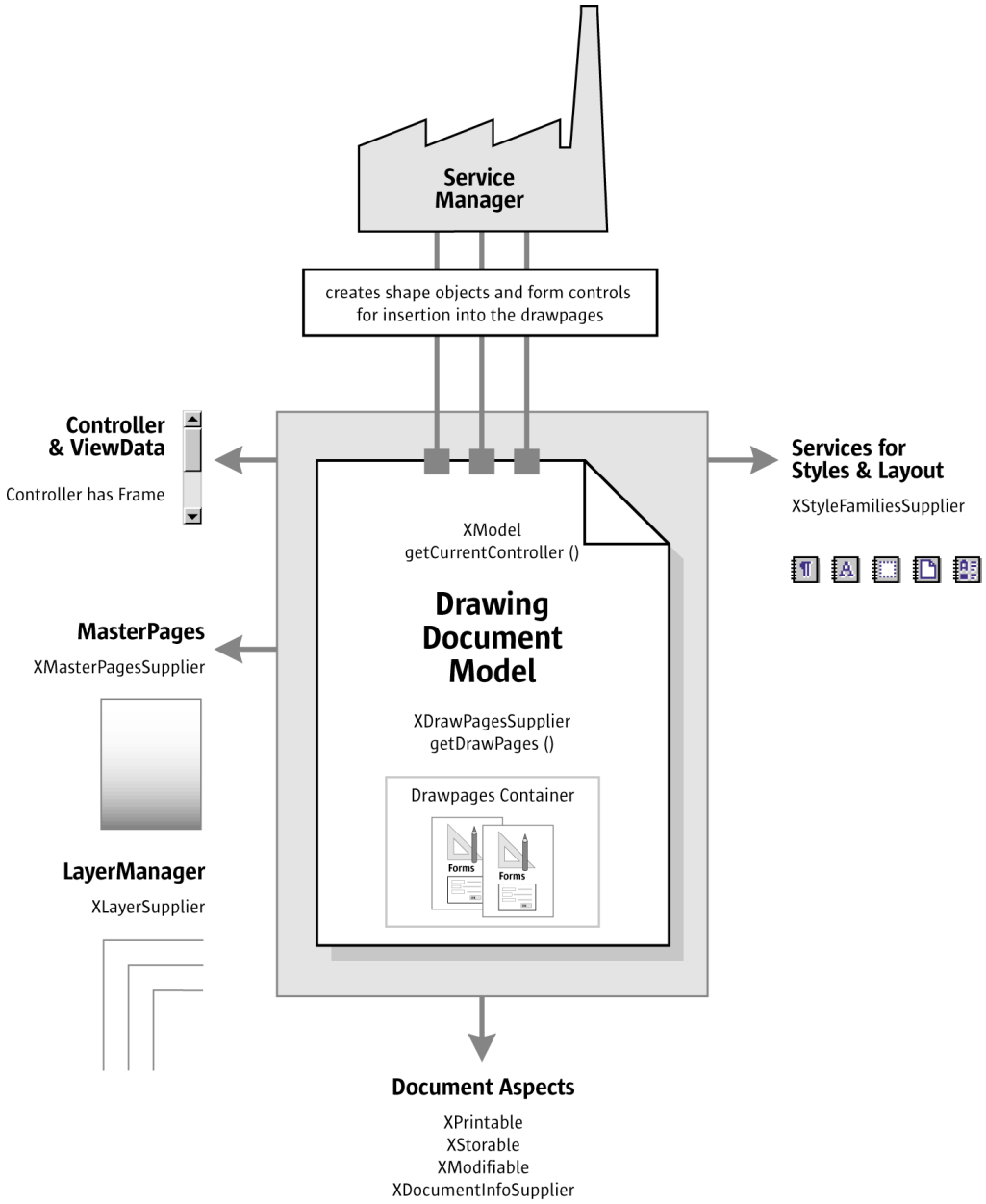


Illustration 129: Drawing Document Overview

In addition to drawing documents, a presentation document has special presentation aspects, which are shown on the lower left of Illustration 132 GraphicExportFilter. There is a presentation supplier to obtain a presentation object, which is used to start and stop presentations, it is possible to edit and run custom presentations and the page layout for presentation handouts is accessible through a handout master supplier.

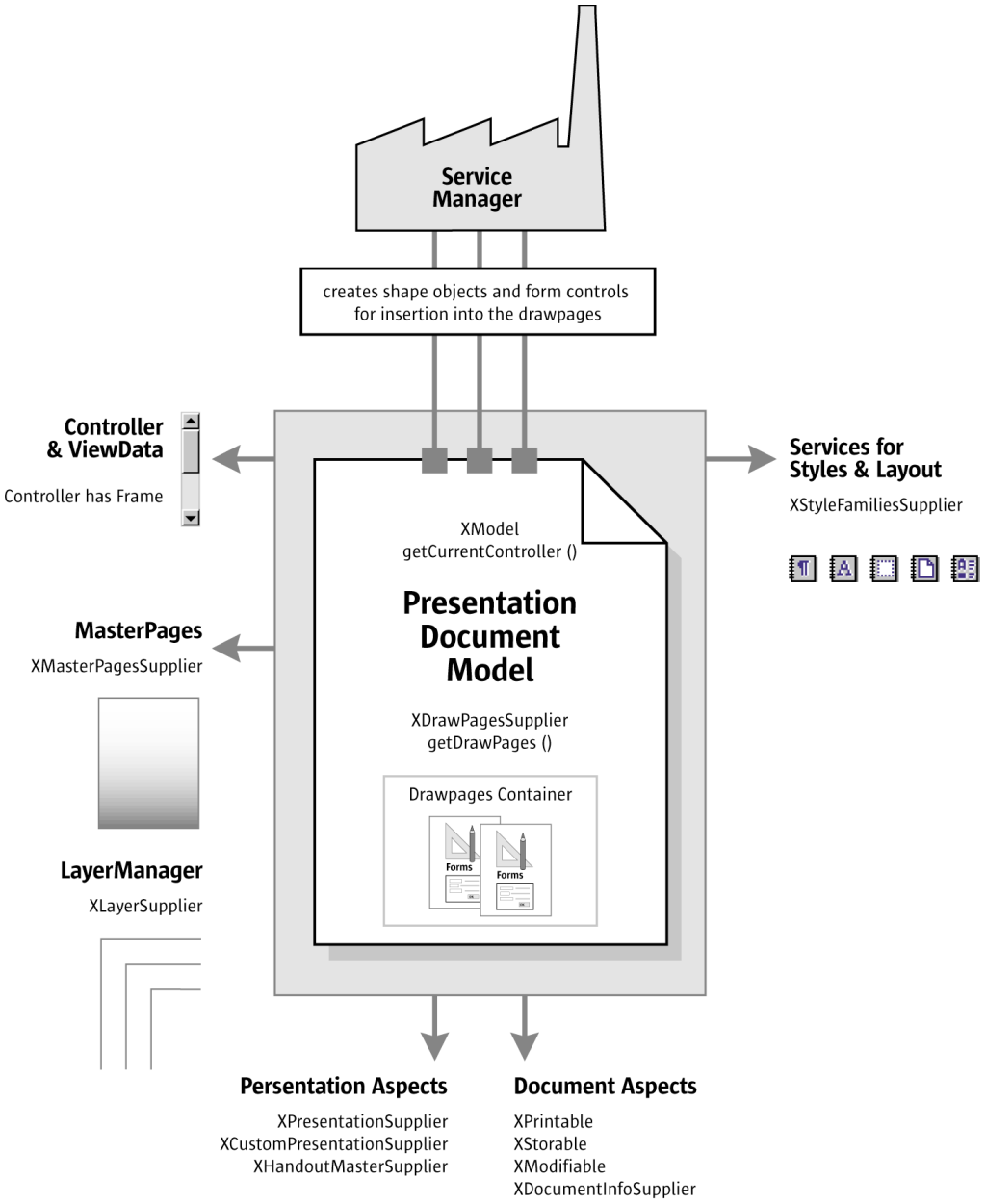


Illustration 130: Presentation Document Overview

9.1.1 Example: Creating a Simple Organizational Chart

The following example creates a simple organizational chart with two levels. It consists of five rectangle shapes and four connectors that connect the boxes on the second level with the root box on the first level.

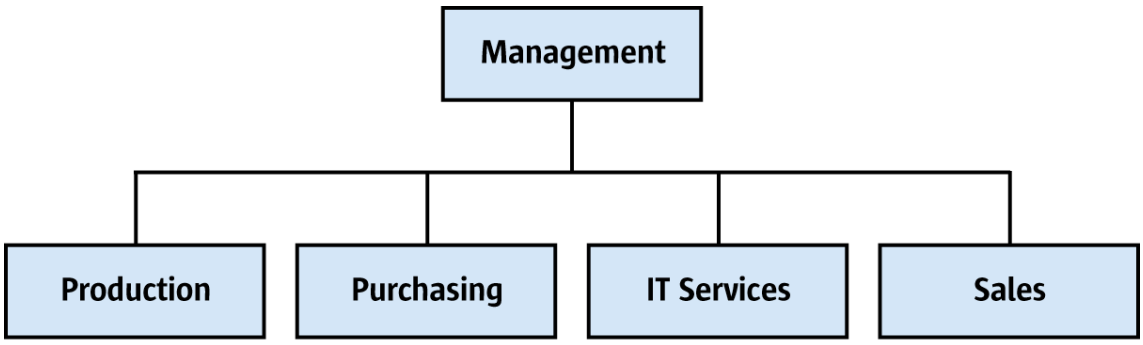


Illustration 131: Sample Organigram

The method `getRemoteServiceManager()` that is used in the example connects to the office. The *2 First Steps* discussed this method. First an empty drawing document is loaded and retrieves the draw page object of slide number 1 to find the page dimensions. Then the organigram data is prepared and the shape sizes are calculated. The shapes are added in a `for` loop that iterates over the organigram data, and connectors are added for all shapes on the second level of the organigram. (Drawing/Organigram.java).

```
public void drawOrganigram() throws java.lang.Exception {

    xRemoteServiceManager = this.getRemoteServiceManager(
        "uno:socket,host=localhost,port=8100;urp;StarOffice.ServiceManager");
    Object desktop = xRemoteServiceManager.createInstanceWithContext(
        "com.sun.star.frame.Desktop", xRemoteContext);
    XComponentLoader xComponentLoader = (XComponentLoader)UnoRuntime.queryInterface(
        XComponentLoader.class, desktop);
    PropertyValue[] loadProps = new PropertyValue[0];
    XComponent xDrawComponent = xComponentLoader.loadComponentFromURL(
        "private:factory/sdraw", "_blank", 0, loadProps);

    // get draw page by index
    com.sun.star.drawing.XDrawPagesSupplier xDrawPagesSupplier =
        (com.sun.star.drawing.XDrawPagesSupplier)
            UnoRuntime.queryInterface(
    com.sun.star.drawing.XDrawPagesSupplier.class, xDrawComponent );
    com.sun.star.drawing.XDrawPages xDrawPages = xDrawPagesSupplier.getDrawPages();
    Object drawPage = xDrawPages.getByIndex(0);
    com.sun.star.drawing.XDrawPage xDrawPage = (com.sun.star.drawing.XDrawPage)
        UnoRuntime.queryInterface(
            com.sun.star.drawing.XDrawPage.class, drawPage);

    // find out page dimensions
    com.sun.star.beans.XPropertySet xPageProps = (com.sun.star.beans.XPropertySet)
        UnoRuntime.queryInterface(
            com.sun.star.beans.XPropertySet.class, xDrawPage);
           int pageWidth = AnyConverter.toInt(xPageProps.getPropertyValue("Width"));
    int pageHeight = AnyConverter.toInt(xPageProps.getPropertyValue("Height"));
    int pageBorderTop = AnyConverter.toInt(xPageProps.getPropertyValue("BorderTop"));
    int pageBorderLeft = AnyConverter.toInt(xPageProps.getPropertyValue("BorderLeft"));
    int pageBorderRight = AnyConverter.toInt(xPageProps.getPropertyValue("BorderRight"));
    int drawWidth = pageWidth - pageBorderLeft - pageBorderRight;
    int horCenter = pageBorderLeft + drawWidth / 2;

    // data for organigram
    String[][] orgUnits = new String[2][4];
    orgUnits[0][0] = "Management";  // level 0
    orgUnits[1][0] = "Production";  // level 1
    orgUnits[1][1] = "Purchasing";  // level 1
    orgUnits[1][2] = "IT Services"; // level 1
    orgUnits[1][3] = "Sales";       // level 1
    int[] levelCount = {1, 4};

    // calculate shape sizes and positions
    int horSpace = 300;
    int verSpace = 3000;
    int shapeWidth = (drawWidth - (levelCount[1] - 1) * horSpace) / levelCount[1];
    int shapeHeight = pageHeight / 20;
    int shapeX = pageWidth / 2 - shapeWidth / 2;
    int levelY = 0;

    com.sun.star.drawing.XShape xStartShape = null;

    // get document factory
```

```
    com.sun.star.lang.XMultiServiceFactory xDocumentFactory = (com.sun.star.lang.XMultiServiceFactory)
        UnoRuntime.queryInterface(
            com.sun.star.lang.XMultiServiceFactory.class, xDrawComponent);

    // creating and adding RectangleShapes and Connectors
    for (int level = 0; level <= 1; level++) {
        levelY = pageBorderTop + 2000 + level * (shapeHeight + verSpace);
        for (int i = levelCount[level] - 1; i > -1; i--) {
            shapeX = horCenter -
                     (levelCount[level] * shapeWidth + (levelCount[level] - 1) * horSpace) / 2 +
                     i * shapeWidth + i * horSpace;
            Object shape = xDocumentFactory.createInstance("com.sun.star.drawing.RectangleShape");
            com.sun.star.drawing.XShape xShape = (com.sun.star.drawing.XShape)
                UnoRuntime.queryInterface(
                    com.sun.star.drawing.XShape.class, shape);
            xShape.setPosition(new com.sun.star.awt.Point(shapeX, levelY));
            xShape.setSize(new com.sun.star.awt.Size(shapeWidth, shapeHeight));
            xDrawPage.add(xShape);

            // set the text
            com.sun.star.text.XText xText = (com.sun.star.text.XText)
                UnoRuntime.queryInterface(
                    com.sun.star.text.XText.class, xShape);
            xText.setString(orgUnits[level][i]);

            // memorize the root shape, for connectors
            if (level == 0 && i == 0)
                xStartShape = xShape;

            // add connectors for level 1
            if (level == 1) {
                Object connector = xDocumentFactory.createInstance(
                    "com.sun.star.drawing.ConnectorShape");
                com.sun.star.drawing.XShape xConnector = (com.sun.star.drawing.XShape)
                    UnoRuntime.queryInterface(
                        com.sun.star.drawing.XShape.class, connector);
                xDrawPage.add(xConnector);
                com.sun.star.beans.XPropertySet xConnectorProps = (com.sun.star.beans.XPropertySet)
                    UnoRuntime.queryInterface(
                        com.sun.star.beans.XPropertySet.class, connector);
                xConnectorProps.setPropertyValue("StartShape", xStartShape);
                xConnectorProps.setPropertyValue("EndShape", xShape);

                // glue point positions: 0=top 1=left 2=bottom 3=right
                xConnectorProps.setPropertyValue("StartGluePointIndex", new Integer(2));
                xConnectorProps.setPropertyValue("EndGluePointIndex", new Integer(0));
            }

        }
    }
}
```

9.2 Handling Drawing Document Files

9.2.1 Creating and Loading Drawing Documents

If a document in OpenOffice.org is required, begin by getting the `com.sun.star.frame.Desktop` service from the service manager. The desktop handles all document components in OpenOffice.org among other things. It is discussed thoroughly in the chapter *6 Office Development*. Office documents are often called *components* because they support the `com.sun.star.lang.XComponent` interface. An `XComponent` is a UNO object that can be disposed explicitly and broadcast an event to other UNO objects when this happens.

The Desktop loads new and existing components from a URL. The desktop has a `com.sun.star.frame.XComponentLoader` interface that has one single method to load and instantiate components from a URL into a frame:

```
com::sun::star::lang::XComponent loadComponentFromURL( [in] string aURL,
                                    [in] string aTargetFrameName,
                                    [in] long nSearchFlags,
                                    [in] sequence< com::sun::star::beans::PropertyValue > aArgs )
```

The parameters in our context are the URL that describes the resource to be loaded, and the load arguments. For the target frame pass in "_blank" and set the search flags to 0. In most cases, you will not want to reuse an existing frame.

The URL can be a file: URL, an http: URL, an ftp: URL or a private: URL. The correct URL format is located in the load URL box at the function bar of OpenOffice.org. For new Draw documents, a special URL scheme is used. The scheme is "private:", followed by "factory" as the hostname and the resource is "sdraw" for OpenOffice.org Draw documents. Thus, for a new Draw document, use "private:factory/sdraw".

The load arguments are described in com.sun.star.document.MediaDescriptor. The properties AsTemplate and Hidden are boolean values and used for programming. If AsTemplate is true, the loader creates a new untitled document from the given URL. If it is false, template files are loaded for editing. If Hidden is true, the document is loaded in the background. This is useful to generate a document in the background without letting the user observe what is happening. For instance, use it to generate a document and print it out without previewing. Refer to *6 Office Development* or other available options.

The introductory example shows how to load a drawing document. This snippet loads a new drawing document in hidden mode:

```
// the method getRemoteServiceManager is described in the chapter First Steps
mxRemoteServiceManager = this.getRemoteServiceManager();

// retrieve the Desktop object, we need its XComponentLoader
Object desktop = mxRemoteServiceManager.createInstanceWithContext(
    "com.sun.star.frame.Desktop", mxRemoteContext);

// query the XComponentLoader interface from the Desktop service
XComponentLoader xComponentLoader = (XComponentLoader)UnoRuntime.queryInterface(
    XComponentLoader.class, desktop);

// define load properties according to com.sun.star.document.MediaDescriptor
// the boolean property Hidden tells the office to open a file in hidden mode
PropertyValue[] loadProps = new PropertyValue[1];
loadProps[0] = new PropertyValue();
loadProps[0].Name = "Hidden";
loadProps[0].Value = new Boolean(true);

/* or simply create an empty array of com.sun.star.beans.PropertyValue structs:
   PropertyValue[] loadProps = new PropertyValue[0]
 */

// load
com.sun.star.lang.XComponent xComponentLoader.loadComponentFromURL(
    "private:factory/sdraw", "_blank", 0, loadProps);
```

9.2.2 Saving Drawing Documents

The normal **File – Save** command for drawing documents can only store the current document in the native OpenOffice.org Draw format and its predecessors. There are other formats that can be stored through the **File – Export** option. This is mirrored in the API. Exporting in the current version of OpenOffice.org Draw and Impress is a different procedure than storing.

Storing

Documents are storable through their interface com.sun.star.frame.XStorable. The *6 Office Development* discusses this in detail. An XStorable implements these operations:

```
boolean hasLocation()
string getLocation()
boolean isReadonly()
void store()
void storeAsURL( [in] string aURL, [in] sequence < com::sun::star::beans::PropertyValue > aArgs)
void storeToURL( [in] string aURL, [in] sequence < com::sun::star::beans::PropertyValue > aArgs)
```

The method names should be evident. The method `storeAsUrl()` is the exact representation of **File – Save As**, that is, it changes the current document location. In contrast, `storeToUrl()` stores a copy to a new location, but leaves the current document URL untouched. There are also store *arguments*. A filter name can be passed that tells OpenOffice.org to use older StarOffice Draw file formats. Exporting is a different matter as shown below. The property needed is `FilterName` which is a string argument that takes filter names defined in the configuration file:

<OfficePath>\share\config\registry\instance\org\openoffice\Office\TypeDetection.xml

In *TypeDetection.xml*, find `<Filter/>` elements, their `cfg:name` attribute contains the required strings for `FilterName`. The correct filter name for StarDraw 5.x files is "StarDraw 5.0". The following is the element in *TypeDetection.xml* that describes the StarDraw 5.0 document filter:

```
<Filter cfg:name="StarDraw 5.0">
    <Installed cfg:type="boolean">true</Installed>
    <UIName cfg:type="string" cfg:localized="true">
        <cfg:value xml:lang="en-US">StarDraw 5.0</cfg:value>
    </UIName>
    <Data cfg:type="string">
        10,draw_StarDraw_50,com.sun.star.drawing.DrawingDocument,,268435559,,5050,,
    </Data>
</Filter>
```

The following method stores a document using this filter:

```
/** Store a document, using the StarDraw 5.0 Filter */
protected void storeDocComponent(XComponent xDoc, String storeUrl) throws java.lang.Exception {
    XStorable xStorable = (XStorable)UnoRuntime.queryInterface(XStorable.class, xDoc);
    PropertyValue[] storeProps = new PropertyValue[1];
    storeProps[0] = new PropertyValue();
    storeProps[0].Name = "FilterName";
    storeProps[0].Value = "StarDraw 5.0";
    xStorable.storeAsURL(storeUrl, storeProps);
}
```

If an empty array of `PropertyValue` structs is passed, the native .sxd format of OpenOffice.org is used.

Exporting

Exporting is not a feature of drawing documents. There is a separate service available from the global service manager for exporting, `com.sun.star.drawing.GraphicExportFilter`. It supports three interfaces: `com.sun.star.document.XFilter`, `com.sun.star.document.XExporter` and `com.sun.star.document.XMimeTypeInfo`.

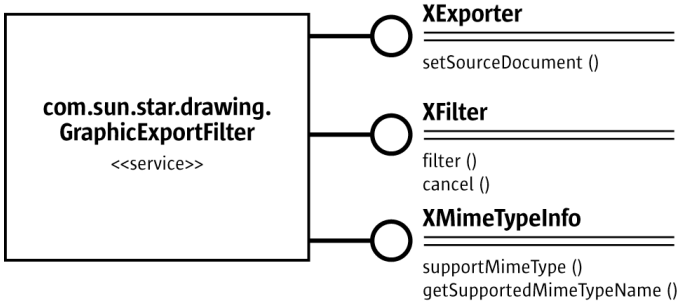


Illustration 132: GraphicExportFilter

Exporting is a simple process. After getting a `GraphicExportFilter` from the `ServiceManager`, use its `XExporter` interface to inform the filter which draw page, shape or shape collection to export.

Method of com.sun.star.document.XExporter:

```
void setSourceDocument ( [in] com::sun::star::lang::XComponent xDoc)
```

The method name `setSourceDocument()` may be confusing. Actually, the method would allow exporting entire documents, however, it is only possible to export draw pages, single shapes or shape collections from a drawing document. Since these objects support the `XComponent` interface, the method specification allows maximum flexibility.

Next, run the method `filter()` at the `XFilter` interface. To interrupt the exporting process, call `cancel()` on the same interface.

Methods of com.sun.star.document.XFilter:

```
boolean filter( [in] sequence< com::sun::star::beans::PropertyValue > aDescriptor)
void cancel()
```

Filter Options

The method `filter()` takes a sequence of `PropertyValue` structs describing the filter parameters. The following properties from the `com.sun.star.document.MediaDescriptor` are supported:

Properties of `com.sun.star.document.MediaDescriptor` supported by GraphicExportFilter	
`MediaType`	Depending on the export filters supported by this component, this is the mime type of the target graphic file. The mime types currently supported are: image/x-MS-bmp application/dxf application/postscript image/gif image/jpeg image/png image/x-pict image/x-pcx image/x-portable-bitmap image/x-portable-graymap image/x-portable-pixmap image/x-cmu-raster image/targa image/tiff image/x-xbitmap image/x-xpixmap image/svg+xml
`FilterName`	This property can been used if no `MediaType` exists with "Windows Metafile" or "Enhanced Metafile". FilterName has to be set to the extension of these graphic formats (WMF, EMF, BMP).
`URL`	The target URL of the file that is created during export.



If necessary, use the interface `XMimeTypeInfo` to get all mime types supported by the `GraphicExport-Filter`. It offers the following methods:

boolean **supportsMimeType**([in] string MimeTypeName)

sequence< string > **getSupportedMimeTypeNames**()

`XMimeTypeInfo` is currently not supported by the `GraphicExportFilter`

The following example exports a draw page `xPage` from a given document xDrawDoc: (Drawing/GraphicExportDemo.java)

```
//get draw pages
com.sun.star.drawing.XDrawPagesSupplier xPageSupplier = (com.sun.star.drawing.XDrawPagesSupplier)
    UnoRuntime.queryInterface(com.sun.star.drawing.XDrawPagesSupplier.class, xDrawDoc);
com.sun.star.drawing.XDrawPages xDrawPages = xPageSupplier.getDrawPages();

// first page
Object page = xDrawPages.getByIndex(0);
com.sun.star.drawing.XDrawPage xPage = (com.sun.star.drawing.XDrawPage)UnoRuntime.queryInterface(
    com.sun.star.drawing.XDrawPage.class, page);

Object GraphicExportFilter = xServiceFactory.createInstance(
    "com.sun.star.drawing.GraphicExportFilter");

// use the XExporter interface to set xPage as source component
// for the GraphicExportFilter
XExporter xExporter = (XExporter)UnoRuntime.queryInterface(
    XExporter.class, GraphicExportFilter );

XComponent xComp = (XComponent)UnoRuntime.queryInterface(XComponent.class, xPage);
xExporter.setSourceDocument(xComp);

// prepare the media descriptor for the filter() method in XFilter
PropertyValue aProps[] = new PropertyValue[2];

aProps[0] = new PropertyValue();
aProps[0].Name = "MediaType";
aProps[0].Value = "image/gif";

// for some graphic formats, e.g. Windows Metafile, there is no Mime type,
// therefore it is also possible to use the property FilterName with
// Filter names as defined in the file TypeDetection.xml (see "Storing")
/* aProps[0].Name = "FilterName";
   aProps[0].Value = "WMF - MS Windows Metafile";
*/

aProps[1] = new PropertyValue();
aProps[1].Name = "URL";
aProps[1].Value = "file:///home/images/page1.gif";

// get XFilter interface and launch the export
XFilter xFilter = (XFilter) UnoRuntime.queryInterface(
    XFilter.class, GraphicExportFilter);
xFilter.filter(aProps);
```

9.2.3 Printing Drawing Documents

Printer and Print Job Settings

Printing is a common office functionality. Refer to Chapter *6 Office Development* for additional information. The Draw document implements the `com.sun.star.view.XPrintable` interface for printing. It consists of three methods:

```
sequence< com::sun::star::beans::PropertyValue > getPrinter()
void setPrinter( [in] sequence< com::sun::star::beans::PropertyValue > aPrinter)
void print( [in] sequence< com::sun::star::beans::PropertyValue > xOptions)
```

To print to the standard printer without settings, use the snippet below with a given document `xDoc`:

```
// query the XPrintable interface from your document
XPrintable xPrintable = (XPrintable)UnoRuntime.queryInterface(XPrintable.class, xDoc);

// create an empty printOptions array
PropertyValue[] printOpts = new PropertyValue[0];

// kick off printing
xPrintable.print(printOpts);
```

There are two groups of properties involved in general printing. The first one is used with `setPrinter()` and `getPrinter()`, and controls the printer, the second one is passed to `print()` and controls the print job.

The method `getPrinter()` returns a sequence of `PropertyValue` structs describing the printer containing the properties specified in the service `com.sun.star.view.PrinterDescriptor`. It comprises the following properties:

Properties of `com.sun.star.view.PrinterDescriptor`	
`Name`	`string` — Specifies the name of the printer queue to be used.
`PaperOrientation`	`com.sun.star.view.PaperOrientation`. Specifies the orientation of the paper.
`PaperFormat`	`com.sun.star.view.PaperFormat`. Specifies a predefined paper size or if the paper size is a user-defined size.
`PaperSize`	`com.sun.star.awt.Size`. Specifies the size of the paper in 1/100 mm.
`IsBusy`	`boolean` — Indicates if the printer is busy.
`CanSetPaperOrientation`	`boolean` — Indicates if the printer allows changes to `PaperOrientation`.
`CanSetPaperFormat`	`boolean` — Indicates if the printer allows changes to `PaperFormat`.
`CanSetPaperSize`	`boolean` — Indicates if the printer allows changes to `PaperSize`.

The `PrintOptions` offer the following choices for a print job:

Properties of `com.sun.star.view.PrintOptions`	
`CopyCount`	`short` — Specifies the number of copies to print.
`FileName`	`string` — If set, specifies the name of a file to print to.
`Collate`	`boolean` — Advises the printer to collate the pages of the copies. If true, a whole document is printed prior to the next copy, otherwise copies for each page are completed together.
`Pages`	`string` — Specifies the pages to print. It has the same format as in the print dialog of the GUI, for example, 1, 3, 4-7, 9.

The following method uses `PrinterDescriptor` and `PrintOptions` to print to a specific printer, and preselect the pages to print:

The following method uses both, `PrinterDescriptor` and `PrintOptions`, to print to a specific printer and preselect the pages to print:

```
protected void printDocComponent(XComponent xDoc) throws java.lang.Exception {
    XPrintable xPrintable = (XPrintable)UnoRuntime.queryInterface(XPrintable.class, xDoc);
    PropertyValue[] printerDesc = new PropertyValue[1];
    printerDesc[0] = new PropertyValue();
    printerDesc[0].Name = "Name";
    printerDesc[0].Value = "5D PDF Creator";

    xPrintable.setPrinter(printerDesc);

    PropertyValue[] printOpts = new PropertyValue[1];
    printOpts[0] = new PropertyValue();
    printOpts[0].Name = "Pages";
    printOpts[0].Value = "1-4,7";

    xPrintable.print(printOpts);
}
```

In Draw documents, one slide is printed as one page on the printer by default. In the example above, slide one through four and slide seven are printed.

Special Print Settings

The printed drawing view (drawings, notes, handout pages, outline), the print quality (color, grayscale), the page options (tile, fit to page, brochure, paper tray) and additional options (page name, date, time, hidden pages) can all be controlled. *9.6.2 Drawing - Overall Document Features - Settings* describes how these settings are used.

9.3 Working with Drawing Documents

9.3.1 Drawing Document

Document Structure

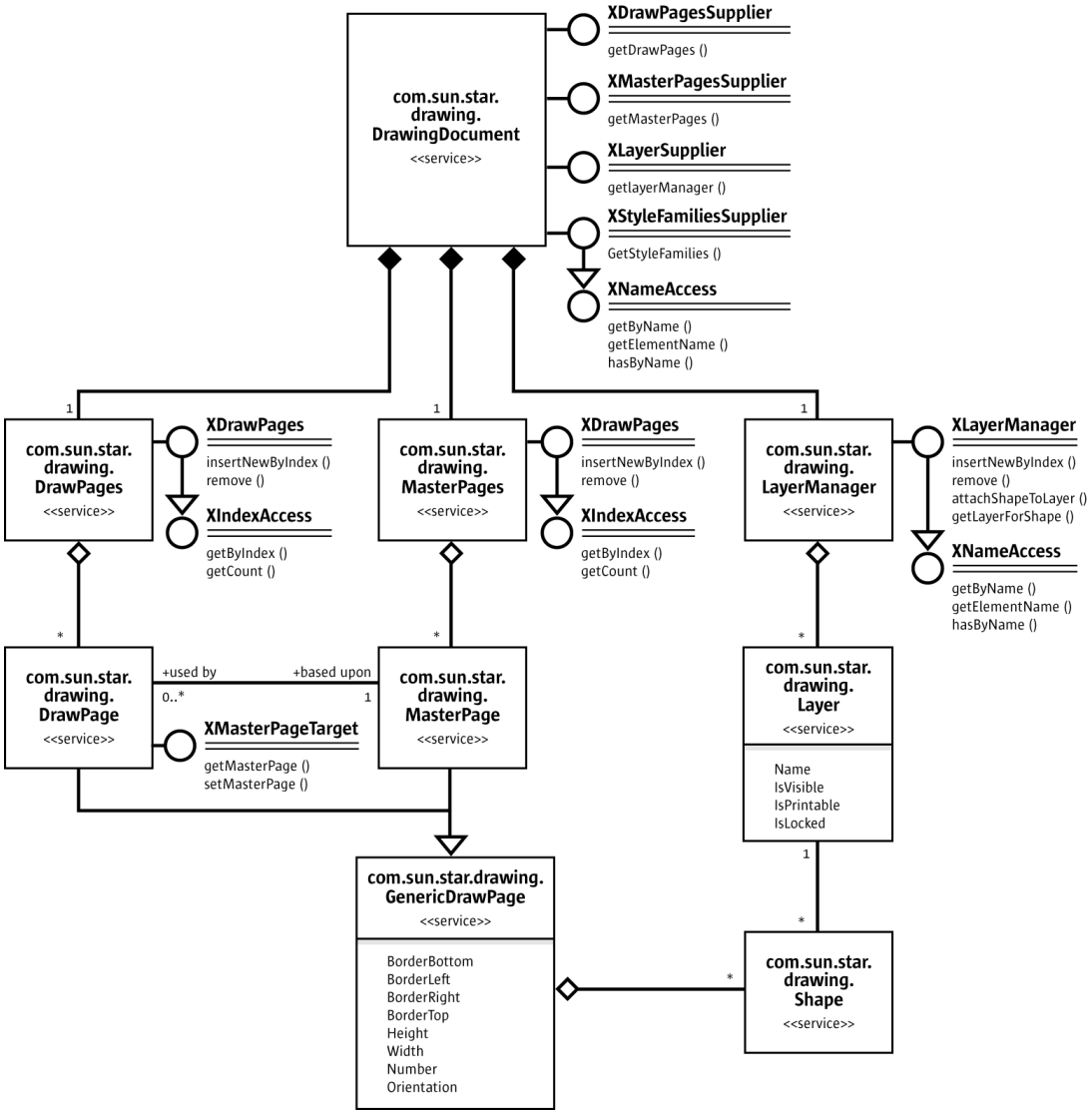


Illustration 133: DrawingDocument Structure

Draw documents maintain their drawing content on draw pages, master pages and layers. If a new draw document is opened, it contains one slide that corresponds to a `com.sun.star.drawing.DrawPage` service. Switching to **Master View** brings up the master page handled by the service `com.sun.star.drawing.MasterPage`. The **Layer View** allows access to layers to structure your drawings. These layers can be controlled through `com.sun.star.drawing.Layer` and `com.sun.star.drawing.LayerManager`.

Page Handling

Draw and Impress documents supply their pages (slides) through the interface `com.sun.star.drawing.XDrawPagesSupplier`. The method `com.sun.star.drawing.XDrawPagesSupplier:getDrawPages()` returns a container of draw pages with a `com.sun.star.drawing.XDrawPages` interface that is derived from `com.sun.star.container.XIndexAccess`. That is, `XDrawPages` allows accessing, inserting and removing pages of a drawing document:

```
type getElementType()
boolean hasElements()
long getCount()
any getByIndex(long Index)

com::sun::star::drawing::XDrawPage insertNewByIndex(long nIndex)
void remove(com::sun::star::drawing::XDrawPage xPage)
```

The example below demonstrates how to access and create draw and master pages. Layers will be described later.

```
XDrawPagesSupplier xDrawPagesSupplier = (XDrawPagesSupplier)UnoRuntime.queryInterface(
    XDrawPagesSupplier.class, xComponent);

// XDrawPages inherits from com.sun.star.container.XIndexAccess
XDrawPages xDrawPages = xDrawPagesSupplier.getDrawPages();

// get the page count for standard pages
int nPageCount = xDrawPages.getCount();

// get draw page by index
XDrawPage xDrawPage = (XDrawPage)xDrawPages.getByIndex(nIndex);

/* create and insert a draw page into the given position,
   the method returns the newly created page
*/
XDrawPage xNewDrawPage = xDrawPages.insertNewByIndex(0);

// remove the given page
xDrawPages.remove( xDrawPage );

/* now repeat the same procedure as described above for the master pages,
   the main difference is to get the XDrawPages from the XMasterPagesSupplier
   interface
*/
XMasterPagesSupplier xMasterPagesSupplier = (XMasterPagesSupplier)UnoRuntime.queryInterface(
    XMasterPagesSupplier.class, xComponent);

XDrawPages xMasterPages = xMasterPagesSupplier.getMasterPages();

// xMasterPages can now be used in the same manner as xDrawPages is used above
```

Each draw page always has *one* master page. The interface `com.sun.star.drawing.XMasterPageTarget` offers methods to get and set the master page that is correlated to a draw page.

```
// query for MasterPageTarget
XMasterPageTarget xMasterPageTarget = (XMasterPageTarget)UnoRuntime.queryInterface(
    XMasterPageTarget.class, xDrawPage);

// now we can get the corresponding master page
XDrawPage xMasterPage = xMasterPageTarget.getMasterPage();

/* this method now sets a new master page,
   it is important to mention that the applied page must be part of the MasterPages
*/
xMasterPageTarget.setMasterPage(xMasterPage);
```

It is possible to copy pages using the interface `com.sun.star.drawing.XDrawPageDuplicator` of drawing or presentation documents.

Methods of `com.sun.star.drawing.XDrawPageDuplicator`:

```
com::sun::star::drawing::XDrawPage duplicate( [in] com::sun::star::drawing::XDrawPage xPage)
```

Pass a draw page reference to the method `duplicate()`. It appends a new draw page at the end of the page list, using the default naming scheme for pages, “slide n”.

Page Partitioning

All units and dimensions are measured in 1/100th of a millimeter. The coordinates are increasing from left to right, and from top to bottom. The upper-left position of a page is (0, 0).

The page size, margins and orientation can be determined using the following properties of a draw page:

Properties of `com.sun.star.drawing.DrawPage`	
`Height`	`long` — Height of the page.
`Width`	`long` — Width of the page.
`BorderBottom`	`long` — Bottom margin of the page.
`BorderLeft`	`long` — Left margin of the page.
`BorderRight`	`long` — Right margin of the page.
`BorderTop`	`long` — Top margin of the page.
`Orientation`	`com.sun.star.view.PaperOrientation`. Determines if the printer output should be turned by 90°. Possible values are: `PORTRAIT` and `LANDSCAPE`.

9.3.2 Shapes

Drawings consist of shapes on draw pages. Shapes are drawing elements, such as rectangles, circles, polygons, and lines. To create a drawing, get a shape by its service name at the `Service-Factory` of a drawing document and add it to the appropriate `DrawPage`.

The code below demonstrates how to create shapes. It consists of a static helper method located in the class `ShapeHelper` and will be used throughout this chapter to create shapes. The parameter `xComponent` must be a reference to a loaded drawing document. The `x`, `y`, `height` and `width` are the desired position and size, and `sShapeType` expects a service name for the shape, such as "com.sun.star.drawing.RectangleShape". The method does not actually insert the shape into a page. It instantiates it and returns the instance to the caller.

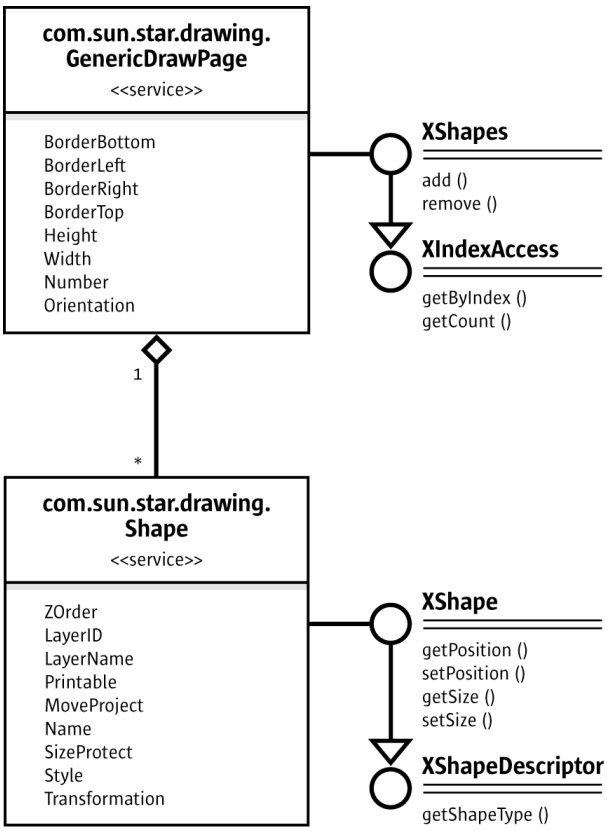


Illustration 134: Shape

The size and position of a shape can be set before adding a shape to a page. After adding the shape, change the shape properties through `com.sun.star.beans.XPropertySet`. (Drawing/Helper.java)

```
public static XShape createShape( XComponent xComponent,
    int x, int y, int width, int height, String sShapeType) throws java.lang.Exception {
    // query the document for the document-internal service factory
    XMultiServiceFactory xFactory = (XMultiServiceFactory)UnoRuntime.queryInterface(
        XMultiServiceFactory.class, xComponent);

    // get the given Shape service from the factory
    Object xObj = xFactory.createInstance(sShapeType);
    Point  aPos = new Point(x, y);
    Size   aSize = new Size(width, height);

    // use its XShape interface to determine position and size before insertion
    xShape = (XShape)UnoRuntime.queryInterface(XShape.class, xObj);
    xShape.setPosition(aPos);
    xShape.setSize(aSize);
    return xShape;
}
```



Notice, the following restrictions: A shape cannot be inserted into multiple pages, and most methods do not work before the shape is inserted into a draw page.

The previously declared method will be used to create a simple rectangle shape with a size of 10 cm x 5 cm that is positioned in the upper-left, and inserted into a drawing page.




```
// query DrawPage for XShapes interface
XShapes xShapes = (XShapes)UnoRuntime.queryInterface(XShapes.class, xDrawPage);

// create the shape
XShape xShape = createShape(xComponent, 0, 0, 10000, 5000, "com.sun.star.drawing.RectangleShape");

// add shape to DrawPage
xShapes.add(xShape);

// set text
XText xText = (XText)UnoRuntime.queryInterface( XText.class, xShape );
xText.setString("My new RectangleShape");

// to be able to set Properties a XPropertySet interface is needed
XPropertySet xPropSet = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, xShape);

xPropSet.setPropertyValue("CornerRadius", new Integer(1000));

xPropSet.setPropertyValue("Shadow", new Boolean(true));
xPropSet.setPropertyValue("ShadowXDistance", new Integer(250));
xPropSet.setPropertyValue("ShadowYDistance", new Integer(250));

// blue fill color
xPropSet.setPropertyValue("FillColor", new Integer(0xC0C0C0));
// black line color
xPropSet.setPropertyValue("LineColor", new Integer(0x000000));

xPropSet.setPropertyValue("Name", "Rounded Gray Rectangle");
```

The UML diagram in Illustration 131 describes all services that are included by the `com.sun.star.drawing.RectangleShape` service and provides an overview of properties that can be used with such a simple shape.

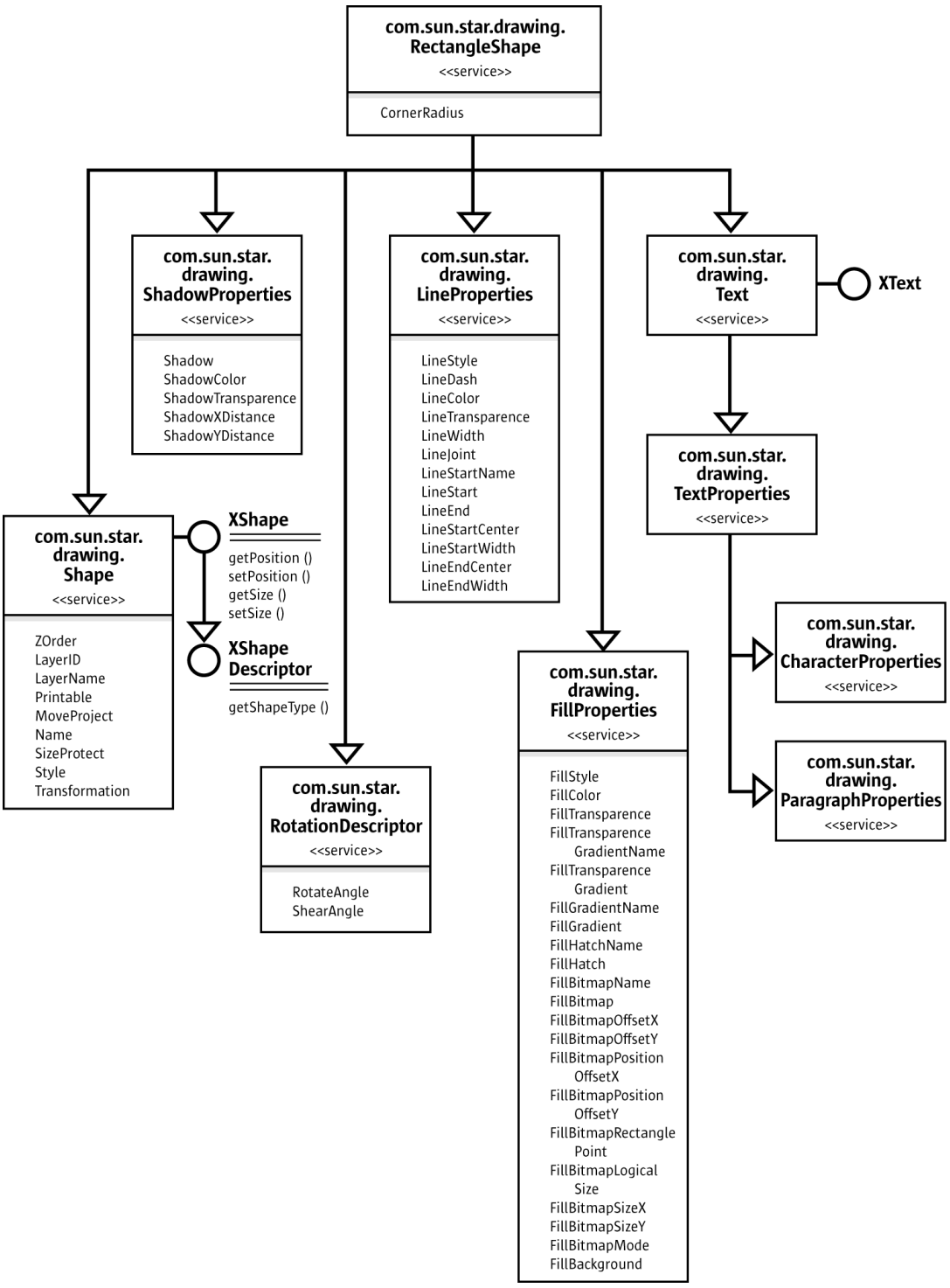


Illustration 135: RectangleShape

Shape Types

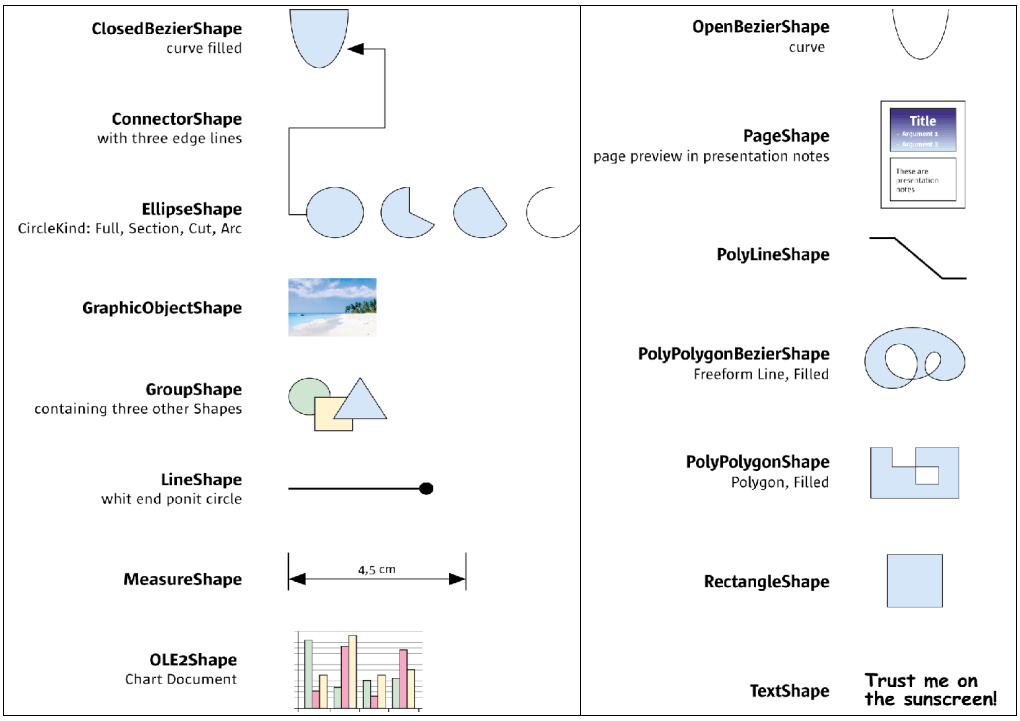


Illustration 136 ShapeTypes

The following table lists all shapes supported in Draw and Impress documents. They come from the `com.sun.star.drawing`. Each shape is based on `com.sun.star.drawing.Shape`. Additionally, there are five services in the module com.sun.star.drawing that most of the shapes have in common:

`com.sun.star.drawing.Text`, `com.sun.star.drawing.LineProperties`, `com.sun.star.drawing.FillProperties` and `com.sun.star.drawing.ShadowProperties` handle shape formatting, whereas `com.sun.star.drawing.RotationDescriptor` controls rotation and shearing. The section *9.3.2 Drawing - Working with Drawing Documents - Shapes - Shape Operations - General Drawing Properties* below discusses shape formatting in more detail. Refer to the section *9.3.2 Drawing - Working with Drawing Documents - Shapes - Shape Operations* for information on rotation and shearing.



The service `com.sun.star.drawing.Text` is different from other Text services. It consists of the service `com.sun.star.drawing.TextProperties` and the interface `com.sun.star.text.XText` that was introduced in the chapter *2 First Steps*. Drawing text does not supports text contents other than paragraphs consisting of character strings.

An *x* denotes which of these services are supported by each shape. The rightmost column shows the services, interfaces and properties that are specific for the various shapes.

ShapeType	***Text***	***LineProperties***	***FillProperties***	***ShadowProperties***	***RotationDescriptor***	**supported services, exported interfaces, properties**
`ClosedBezierShape`	x	x	x	x	x	included service: `com.sun.star.drawing.PolyPolygonBezierDescriptor`
`[IDLScom.sun.star.drawing:ConnectorShape]`	x	x		x	x	included service: `com.sun.star.drawing.ConnectorProperties` properties: `com.sun.star.drawing.XShape` StartShape `com.sun.star.drawing.XShape` EndShape `com.sun.star.awt.Point` StartPosition `com.sun.star.awt.Point` EndPosition long StartGluePointIndex long EndGluePointIndex long EdgeLine1Delta long EdgeLine2Delta long EdgeLine3Delta
`ControlShape`						exported interface: `com.sun.star.drawing.XControlShape`
`EllipseShape`	x	x	x	x	x	properties: `com.sun.star.drawing.CircleKind` CircleKind long CircleStartAngle long CircleEndAngle
`GraphicObjectShape`	x			x	x	properties: string GraphicURL string GraphicStreamURL short AdjustLuminance short AdjustContrast short AdjustRed short AdjustGreen short AdjustBlue double Gamma short Transparency `com.sun.star.drawing.ColorMode` GraphicColorMode optional properties: `com.sun.star.awt.XBitmap` GraphicObjectFillBitmap `com.sun.star.container.XIndexContainer` ImageMap
`GroupShape`						exported interfaces: `com.sun.star.drawing.XShapeGroup` `com.sun.star.drawing.XShapes`
`LineShape`	x	x		x	x	included service: `com.sun.star.drawing.PolyPolygonDescriptor`
`MeasureShape`	x	x		x	x	included service: `com.sun.star.drawing.MeasureProperties` properties: `com.sun.star.awt.Point` StartingPosition `com.sun.star.awt.Point` EndPosition

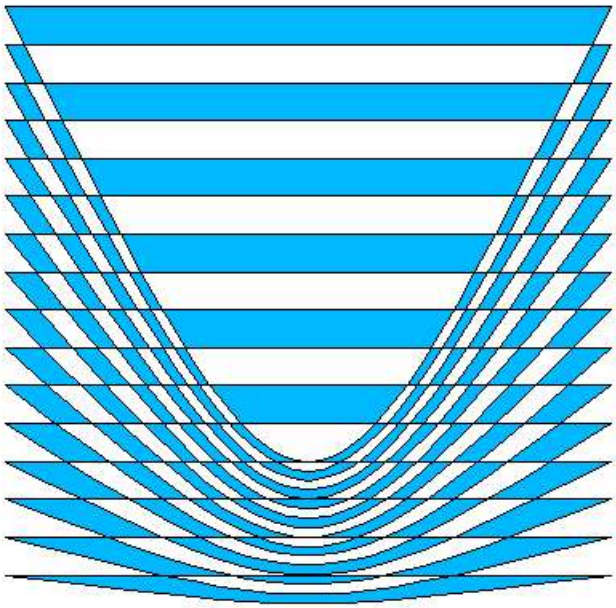
ShapeType	***Text***	***LineProperties***	***FillProperties***	***ShadowProperties***	***RotationDescriptor***	**supported services,** **exported interfaces,** **properties**
`OLE2Shape`						properties: string CLSID readonly properties: `com.sun.star.frame.XModel` Model boolean IsInternal
`OpenBezierShape`	x	x		x	x	included service: `com.sun.star.drawing.PolyPolygonBezierDescriptor`
`PageShape`						
`PolyLineShape`	x	x		x	x	included service: `com.sun.star.drawing.PolyPolygonDescriptor`
`PolyPolygonBezierShape`	x	x	x	x	x	included service: `com.sun.star.drawing.PolyPolygonBezierDescriptor`
`PolyPolygonShape`	x	x	x	x	x	included service: `com.sun.star.drawing.PolyPolygonDescriptor`
`RectangleShape`	x	x	x	x	x	properties: long CornerRadius
`TextShape`	x	x	x	x	x	properties: long CornerRadius
`PluginShape`						properties: string PluginMimeType string PluginURL sequence< `com.sun.star.beans.PropertyValue` > PluginCommands

Bezier Shapes

Draw supports three different kinds of Bezier curves: `OpenBezierShape`, `ClosedBezierShape` and `PolyPolygonBezierShape`. They are all controlled by `com.sun.star.drawing.PolyPolygonBezierDescriptor` which is made up of the following properties:

Properties of `com.sun.star.drawing.PolyPolygonDescriptor`	
`PolygonKind`	[readonly] `com.sun.star.drawing.PolygonKind`. Type of the polygon. Possible values are: `LINE` for a `LineShape`. `POLY` for a `PolyPolygonShape`. `PLIN` for a `PolyLineShape`. `PATHLINE` for an `OpenBezierShape`. `PATHFILL` for a `ClosedBezierShape`.
`PolyPolygonBezier`	struct `com.sun.star.drawing.PolyPolygonBezierCoords`. These are the bezier points of the polygon. The struct members are `Coordinates` and `Flags`, which are both sequences of sequences. The Coordinates sequence contains `com.sun.star.awt.Point` structs and the `Flags` sequence contains com::`com.sun.star.drawing.PolygonFlags` enums. Point members are X and Y. Possible `PolygonFlags` values are: • `NORMAL` the point is normal, from the curve discussion view. • `SMOOTH` the point is smooth, the first derivation from the curve discussion view. • `CONTROL` the point is a control point, to control the curve from the user interface. • `SYMMETRIC` the point is symmetric, the second derivation from the curve discussion view.
`Geometry`	`com.sun.star.drawing.PolyPolygonBezierCoords`. These are the untransformed bezier coordinates of the polygon. The property has the same type as `PolyPolygonBezier`.

The next Java example will demonstrate how to create a `ClosedBezierShape` that looks like the following picture. (Drawing/DrawingDemo.java)



```
XShape xPolyPolygonBezier = createShape( xComponent, 0, 0, 0, 0,
    "com.sun.star.drawing.ClosedBezierShape");

// take care of the fact that the shape must have been added
// to the page before it is possible to apply changes
XShapes xShapes = (XShapes)UnoRuntime.queryInterface( XShapes.class, xDrawPage);
xShapes.add(xPolyPolygonBezier);

// now it is possible to edit the PropertySet
XPropertySet xShapeProperties = (XPropertySet)UnoRuntime.queryInterface(
    XPropertySet.class, xPolyPolygonBezier);

// The following values are exemplary and provokes that a PolyPolygon of
// sixteen single polygons containing four points each is created. The
// PolyPolygon total point count will be 64.
// If control points are used they are allowed to appear as pair only,
// before and after such pair has to be a normal point.

// A bezier point sequence may look like
// this (n=normal, c=control) : n c c n c c n n c c n

int nPolygonCount   = 16;
int nPointCount     = 4;
int nWidth          = 10000;
int nHeight         = 10000;

PolyPolygonBezierCoords aCoords = new PolyPolygonBezierCoords();

// allocating the outer sequence
aCoords.Coordinates = new Point[nPolygonCount][];
aCoords.Flags       = new PolygonFlags[nPolygonCount][];
int i, n, nY;

// fill the inner point sequence now
for (nY = 0, i = 0; i < nPolygonCount; i++, nY += nHeight / nPolygonCount) {
    // create a polygon using two normal and two control points
    // allocating the inner sequence

    Point[]          pPolyPoints = new Point[nPointCount];
    PolygonFlags[]   pPolyFlags  = new PolygonFlags[nPointCount];

    for (n = 0; n < nPointCount; n++)
        pPolyPoints[n] = new Point();

    pPolyPoints[0].X = 0;
    pPolyPoints[0].Y = nY;
    pPolyFlags [0] = PolygonFlags.NORMAL;

    pPolyPoints[1].X = nWidth / 2;
    pPolyPoints[1].Y = nHeight;
    pPolyFlags[1] = PolygonFlags.CONTROL;

    pPolyPoints[2].X = nWidth / 2;
```

```
        pPolyPoints[2].Y = nHeight;
        pPolyFlags [2] = PolygonFlags.CONTROL;

        pPolyPoints[3].X = nWidth;
        pPolyPoints[3].Y = nY;
        pPolyFlags [3] = PolygonFlags.NORMAL;

        aCoords.Coordinates[i] = pPolyPoints;
        aCoords.Flags[i] = pPolyFlags;
    }
    try {
        xShapeProperties.setPropertyValue("PolyPolygonBezier", aCoords);
    } catch (Exception ex)
    {
    }
```

Shape Operations

Moving and Scaling

Moving and scaling of a shape can be done by using the corresponding methods `getPosition()`, `setPosition()`, `getSize()` and `setSize()` of the `com.sun.star.drawing.XShape` interface:

```
string getShapeType()
com::sun::star::awt::Point getPosition()
void setPosition( [in] com::sun::star::awt::Point aPosition)
com::sun::star::awt::Size getSize()
void setSize( [in] com::sun::star::awt::Size aSize)
```

`Point` and `Size` are IDL structs. In Java, these structs are mapped to classes with constructors that take values for the struct members. Therefore, when `new` is used to instantiate these classes, the coordinates and dimensions are passed to initialize the class members X, Y, Width and Height.

Rotating and Shearing

Most shapes, except OLE and group objects, can be *rotated* and *sheared*. All of these objects include the `com.sun.star.drawing.RotationDescriptor` service that has the properties `RotateAngle` and `ShearAngle`.

Setting the `com.sun.star.drawing.RotationDescriptor` rotates or shears a shape:

Properties of `com.sun.star.drawing.RotationDescriptor`	
`RotateAngle`	`long` — This is the angle for rotation of this shape in 1/100th of a degree. The shape is rotated counter-clockwise around the center of the bounding box.
`ShearAngle`	`long` — This is the amount of shearing for this shape in 1/100th of a degree. The shape is sheared clockwise around the center of the bounding box.



Notice that the rotation works counter-clockwise, while shearing works clockwise.

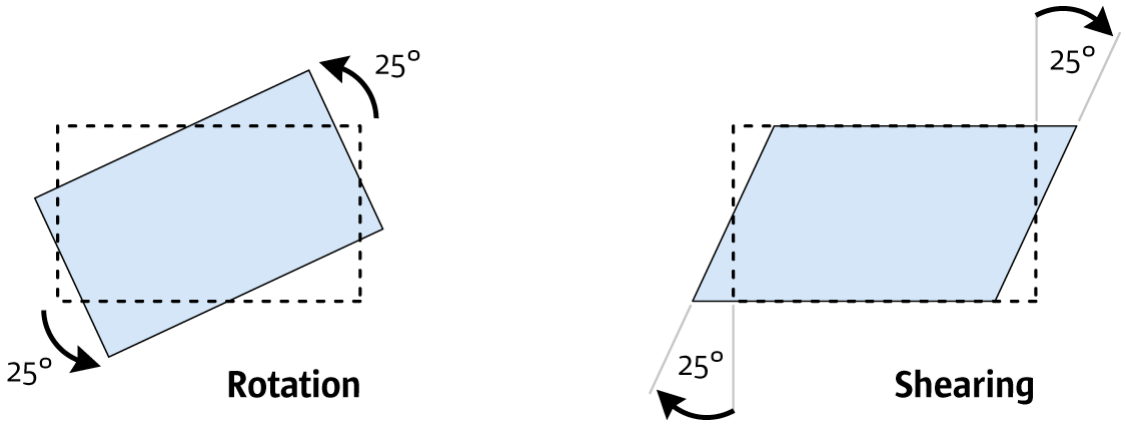


Illustration 137 Rotation and Shearing by 25 degrees

The following example shows how a shape can be rotated by 25 degrees counterclockwise:

```
// xShape will be rotated by 25 degrees
    XPropertySet xPropSet = (XPropertySet)UnoRuntime.queryInterface(
        XPropertySet.class, xShape );
    xPropSet.setPropertyValue( "RotateAngle", new Integer( 2500 ) );
```

Transforming

Changing the size, rotation and shearing of an object can be done by using the transformation mechanism provided by OpenOffice.org. The matrix of our API is a standard homogenous 3x3 matrix that may be used together with the java.awt.geom.AffineTransform class from Java. The transformation received describes the actual values of the transformations as a linear combination of the single matrices. The basic object without transformation has a size of (1, 1) and a position of (0, 0), and is not rotated or sheared. Thus, to transform an object get its matrix and multiply from the left side to influence the current appearance. To set the whole transformation directly, build a combined matrix of the single values mentioned above and apply it to the object. (Drawing/ObjectTransformationDemo.java)

```
    XPropertySet xPropSet = (XPropertySet)UnoRuntime.queryInterface( XPropertySet.class, xShape );

    // take the current tranformation matrix
    HomogenMatrix3 aHomogenMatrix3 = (HomogenMatrix3)xPropSet.getPropertyValue("Transformation");

    java.awt.geom.AffineTransform aOriginalMatrix = new java.awt.geom.AffineTransform(
        aHomogenMatrix3.Line1.Column1, aHomogenMatrix3.Line2.Column1,
        aHomogenMatrix3.Line1.Column2, aHomogenMatrix3.Line2.Column2,
        aHomogenMatrix3.Line1.Column3, aHomogenMatrix3.Line2.Column3 );


    // rotate the object by 15 degrees
    AffineTransform aNewMatrix1 = new AffineTransform();
    aNewMatrix1.setToRotation(Math.PI /180 * 15);
    aNewMatrix1.concatenate(aOriginalMatrix);

    // and translate the object by 2cm on the x-axis
    AffineTransform aNewMatrix2 = new AffineTransform();
    aNewMatrix2.setToTranslation(2000, 0);
    aNewMatrix2.concatenate(aNewMatrix1);

    double aFlatMatrix[] = new double[6];
    aNewMatrix2.getMatrix(aFlatMatrix);

    // convert the flatMatrix to our HomogenMatrix3 structure
    aHomogenMatrix3.Line1.Column1 = aFlatMatrix[0];
    aHomogenMatrix3.Line2.Column1 = aFlatMatrix[1];
    aHomogenMatrix3.Line1.Column2 = aFlatMatrix[2];
    aHomogenMatrix3.Line2.Column2 = aFlatMatrix[3];
    aHomogenMatrix3.Line1.Column3 = aFlatMatrix[4];
    aHomogenMatrix3.Line2.Column3 = aFlatMatrix[5];

    xPropSet.setPropertyValue("Transformation", aHomogenMatrix3);
```

Ordering

The property `ZOrder` of the `com.sun.star.drawing.Shape` service defines the order a shape is drawn. That is, if there are many shapes on a page, the shape that has the lowest `ZOrder` value is drawn first, and the shape that has the highest `ZOrder` is drawn last. By using this property it is possible to bring an object to the back or front of a page. It is also possible to switch the order of two shapes as demonstrated in the following example: (Drawing/ChangeOrderDemo.java)

```
XPropertySet xPropSet1 = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, xShape1);
XPropertySet xPropSet2 = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, xShape2);

// get current positions
int nOrderOfShape1 = ((Integer)xPropSet1.getPropertyValue("ZOrder")).intValue();
int nOrderOfShape2 = ((Integer)xPropSet2.getPropertyValue("ZOrder")).intValue();

// set new positions
xPropSet1.setPropertyValue("ZOrder", new Integer(nOrderOfShape2));
xPropSet2.setPropertyValue("ZOrder", new Integer(nOrderOfShape1));
```

Grouping, Combining and Binding

The DrawPage plays an important role for the handling of multiple shapes. It has three interfaces for this purpose. Its interface `com.sun.star.drawing.XShapeGrouper` is used to create a group shape from a `ShapeCollection` and ungroup existing groups.

Methods of `com.sun.star.drawing.XShapeGrouper`	
`group()`	Parameter: `com.sun.star.drawing.XShapes xShapes` Groups the shapes inside a collection. They must all be inserted into the same `GenericDrawPage` . Returns a recently created `GroupShape` that contains all shapes from `xShapes`, and is also added to the `GenericDrawPage` of the Shapes in `xShapes`.
`ungroup()`	Parameter: `com.sun.star.drawing.XShapeGroup` Ungroups a given `GroupShape`. Moves all Shapes from this `GroupShape` to the parent `XShapes` of the `GroupShape`. The `GroupShape` is then removed from the `GenericDrawPage` and disposed.

The example below creates a group using the `com.sun.star.drawing.XShapeGrouper` interface. For this purpose, the shapes that are to be grouped have to be added to a `com.sun.star.drawing.ShapeCollection` that is created by the `com.sun.star.lang.XMultiServiceFactory` of the *global* service manager. It is a container of shapes that is accessed using the interface `com.sun.star.drawing.XShapes`. The following example accesses the `XShapes` interface of the `DrawPage` to locate two shapes on the `DrawPage`, and uses the `XShapes` interface of the `ShapeCollection` to add these shapes to the `ShapeCollection`. Finally, it employs the `XShapeGrouper` interface of the `DrawPage` to move the shapes from the `ShapeCollection` into a new `GroupShape`. (Drawing/ControlAndSelectDemo)

```
/* try to group the first two objects of the drawpage */

// create a container that will receive the
// shapes that are to be grouped
Object xObj = xMultiServiceFactory.createInstance("com.sun.star.drawing.ShapeCollection");
XShapes xToGroup = (XShapes)UnoRuntime.queryInterface(XShapes.class, xObj);

// query for the shape collection of xDrawPage
XShapes xShapes = (XShapes)UnoRuntime.queryInterface(XShapes.class, xDrawPage);

// test if the shape collection of the page has at least two shapes
if (xShapes.getCount() >= 2) {
```

```
        // collect shapes we want to group
        xToGroup.add(xShapes.getByIndex(0));
        xToGroup.add(xShapes.getByIndex(1));

        // now group the shapes we have collected by using the XShapeGrouper
        XShapeGrouper xShapeGrouper = (XShapeGrouper)UnoRuntime.queryInterface(
            XShapeGrouper.class, xDrawPage);
        xShapeGrouper.group(xToGroup);
    }
```

The service `com.sun.star.drawing.GroupShape` includes `com.sun.star.drawing.Shape` and supports two additional interfaces:

- `com.sun.star.drawing.XShapes` is used to access the shapes in the group.
- `com.sun.star.drawing.XShapeGroup` handles access to the group.

The interface `XShapes` inherits from `com.sun.star.container.XIndexAccess`, and introduces `add()` and `remove()`. It contains the following methods:

type **getElementType**()

```
boolean hasElements()
long getCount()
any getByIndex( [in] long Index)
void add( [in] com::sun::star::drawing::XShape xShape)
void remove( [in] com::sun::star::drawing::XShape xShape)
```

Methods of `com.sun.star.drawing.XShapeGroup`:

```
string getShapeType()
com::sun::star::awt::Point getPosition()
void setPosition( [in] com::sun::star::awt::Point aPosition)
com::sun::star::awt::Size getSize()
void setSize( [in] com::sun::star::awt::Size aSize)
```

It is also possible to create GroupShapes directly without using the XShapeGrouper interface. The following code demonstrates the creation of a `com.sun.star.drawing.GroupShape` that takes up three other shapes. (Drawing/DrawingDemo.java)

```
    // create a group shape first. The size and position does not matter, because
    // it depends to the position and size of objects that will be inserted later
    XShape xGroup = createShape(xComponent, 0, 0, 0, 0, "com.sun.star.drawing.GroupShape");

    // before it is possible to insert shapes,
    // the group shape must have been added to the page
    XShapes xShapes = (XShapes)UnoRuntime.queryInterface(XShapes.class, xDrawPage);
    xShapes.add(xGroup);

    // query for the XShapes interface, which will take our new shapes
    XShapes xShapesGroup = (XShapes)UnoRuntime.queryInterface(XShapes.class, xGroup);

    // new shapes can be inserted into the shape collection directly
    xShapesGroup.add( createShape(xComponent, 1000, 1000, 2000, 4000,
        "com.sun.star.drawing.EllipseShape"));
    xShapesGroup.add( createShape(xComponent, 8000, 8000, 2000, 2000,
        "com.sun.star.drawing.EllipseShape"));
    xShapesGroup.add( createShape(xComponent, 2000, 3000, 7000, 6000,
        "com.sun.star.drawing.LineShape"));
```

The interface `com.sun.star.drawing.XShapeCombiner` combines shapes and is equivalent to **Modify – Combine** in the user interface.

Methods of `com.sun.star.drawing.XShapeCombiner`	
`combine()`	Parameter: `com.sun.star.drawing.XShapes` Combines shapes. The shapes inside this container are converted to `PolyPolygonBezierShapes` and are than combined into one `PolyPolygonBezierShape`. The shapes in `xShape` are removed from the `GenericDrawPage` and disposed. Returns a recently created `PolyPolygonBezierShape` that contains all the converted `PolyPolygonBezierShapes` combined. It is also added to the `GenericDrawPage` of the source `Shapes`.
`split()`	Parameter: `com.sun.star.drawing.XShape` Splits shapes. The Shape is converted to a `PolyPolygonBezierShape` and then split into several `PolyPolygonBezierShapes`. The shape s in `xShapeare` removed from the `GenericDrawPage` and disposed.

The draw page interface `com.sun.star.drawing.XShapeBinder` draws a *connection line* between the ending point of a line shape (or curve) to the starting point of another line shape (or curve), merging the connected lines into a single shape object. This function corresponds to **Modify – Connect** in the user interface. It works for area shapes as well, but the connection line usually can not resolve them.

Methods of `com.sun.star.drawing.XShapeBinder`	
`bind()`	Parameter: `com.sun.star.drawing.XShapes` binds shapes together. A container with shapes that will be bound together. All shapes are converted to a `PolyPolygonBezierShape` and the lines are connected. The Shapes in `xShape` are removed from the `GenericDrawPage` and disposed. Returns a recently created `PolyPolygonBezierShape` that contains all line segments from the supplied Shapes. It is also added to the `GenericDrawPage` of the source Shapes.
`unbind()`	Parameter: `com.sun.star.drawing.XShape` breaks a shape into its line segments. The given shape will be converted to a `PolyPolygonBezierShape` and the line segments of this shape are used to create new `PolyPolygonBezierShape` shapes. The original shape is removed from its `GenericDrawPage` and disposed.

General Drawing Properties

This chapter introduces the relevant drawing attributes provided by services, such as `com.sun.star.drawing.LineProperties`, `com.sun.star.drawing.FillProperties` and `com.sun.star.drawing.TextProperties`. The service is described by listing all its properties,

followed by an example that uses and explains some of the properties. Each of the following Java examples assumes an already existing valid shape `xShape` that has already been inserted into the page.

Colors are given in Hex ARGB format, a four-byte value containing the alpha, red, green and blue components of a color in the format 0xAARRGGBB. The leading component can be omitted if it is zero. The hex format 0xFF0000 is light red, 0xFF00 is green, and 0xFF is blue.

Angles must be given in steps of 1/100th of a degree.

Measures, such as line widths and lengths are given in 100th of a millimeter.

Properties provided by the service :

Properties of `com.sun.star.drawing.LineProperties`	
`LineStyle`	`com.sun.star.drawing.LineStyle`. This enumeration selects the style of the line.
`LineDash`	`com.sun.star.drawing.LineDash`. This enumeration selects the dash of the line
`LineColor`	long — Color of the line.
`LineTransparence`	`short` — Degree of transparency.
`LineWidth`	`long` — Width of the line in 1/100th of a millimeter.
`LineJoint`	`com.sun.star.drawing.LineJoint`. Rendering of joints between thick lines.
`LineStartName`	`[optional] string` — Name of the line that starts poly polygon bezier.
`LineStart`	`[optional] com.sun.star.drawing.PolyPolygonBezierCoords`. Line starts in the form of a poly polygon bezier.
`LineEnd`	`[optional] com.sun.star.drawing.PolyPolygonBezierCoords`. Line ends in the form of a poly polygon bezier.
`LineStartCenter`	`[optional] boolean` — If true, the line starts from the center of the polygon.
`LineStartWidth`	`[optional] long` — Width of the line start polygon.
`LineEndCenter`	`[optional] boolean` — If true, the line ends in the center of the polygon.
`LineEndWidth`	[optional] long — Width of the line end polygon.

(Drawing/FillAndLineStyleDemo.java)

```
/* create a blue line with dashes and dots */
XPropertySet xPropSet = (XPropertySet)UnoRuntime.queryInterface(
    XPropertySet.class, xShape );
xPropSet.setPropertyValue( "LineStyle", LineStyle.DASH );
LineDash aLineDash = new LineDash();
aLineDash.Dots = 3;
aLineDash.DotLen = 150;
aLineDash.Dashes = 3;
aLineDash.DashLen = 300;
aLineDash.Distance = 150;
xPropSet.setPropertyValue( "LineDash", aLineDash );
xPropSet.setPropertyValue( "LineColor", new Integer( 0x0000ff ) );
xPropSet.setPropertyValue( "LineWidth", new Integer( 200 ) );
```

Properties of `com.sun.star.drawing.FillProperties`	
`FillStyle`	`com.sun.star.drawing.FillStyle`. This enumeration selects the style that the area is filled with.
`FillColor`	`long` — If the `FillStyle` is set to `SOLID`, this is the color used.

Properties of com.sun.star.drawing.FillProperties	
FillTransparence	short — The transparency of the filled area in percent.
FillTransparenceGradientName	string — This is the name of the transparent gradient style used if a gradient is used for transparency, or it is empty. This style is used to set the name of a transparent gradient style contained in the document.
FillTransparenceGradient	[optional]com.sun.star.awt.Gradient. Transparency of the fill area as a gradient.
FillGradientName	string — If the FillStyle is set to GRADIENT, this is the name of the fill gradient style used.
FillGradient	[optional]com.sun.star.awt.Gradient. If the FillStyle is set to GRADIENT, this describes the gradient used.
FillHatchName	string — If the FillStyle is set to GRADIENT, this is the name of the fill hatch style used.
FillHatch	[optional]com.sun.star.drawing.Hatch. If the FillStyle is set to HATCH, this describes the hatch used.
FillBitmapName	string — If the FillStyle is set to BITMAP, this is the name of the fill bitmap style used.
FillBitmap	[optional]com.sun.star.awt.XBitmap. If the FillStyle is set to BITMAP, this is the bitmap used.
FillBitmapURL	[optional] string. If the FillStyle is set to BITMAP, this is a URL to the bitmap used.
FillBitmapOffsetX	short — Horizontal offset where the tile starts.
FillBitmapOffsetY	short — Vertical offset where the tile starts. It is given in percent in relation to the width of the bitmap.
FillBitmapPositionOffsetX	short — Every second line of tiles is moved the given percent of the width of the bitmap.
FillBitmapPositionOffsetY	short — Every second row of tiles is moved the given percent of the width of the bitmap.
FillBitmapRectanglePoint	com.sun.star.drawing.RectanglePoint. The RectanglePoint specifies the position inside of the bitmap to use as the top-left position for rendering.
FillBitmapLogicalSize	boolean — Specifies if the size is given in percentage or as an absolute value.
FillBitmapSizeX	long — Width of the tile for filling.
FillBitmapSizeY	long — Height of the tile for filling.
FillBitmapMode	com.sun.star.drawing.BitmapMode. Enumeration selects how an area is filled with a single bitmap.
FillBackground	boolean — If true , the transparent background of a hatch filled area is drawn in the current background color.

(Drawing/FillAndLineStyleDemo.java)

```
    /* apply a gradient fill style that goes from top left to bottom
       right and is changing its color from green to yellow */

    XPropertySet xPropSet = (XPropertySet)UnoRuntime.queryInterface(
        XPropertySet.class, xShape );

    xPropSet.setPropertyValue( "FillStyle", FillStyle.GRADIENT );
    Gradient aGradient = new Gradient();
    aGradient.Style = GradientStyle.LINEAR;
    aGradient.StartColor = 0x00ff00;
    aGradient.EndColor = 0xffff00;
```

```
    aGradient.Angle = 450;
    aGradient.Border = 0;
    aGradient.XOffset = 0;
    aGradient.YOffset = 0;
    aGradient.StartIntensity = 100;
    aGradient.EndIntensity = 100;
    aGradient.StepCount = 10;
    xPropSet.setPropertyValue( "FillGradient", aGradient );
```

Properties of `com.sun.star.drawing.TextProperties`	
`IsNumbering`	[optional] `boolean` — If `true`, numbering is on for the text of this shape.
`NumberingRules`	[optional] `com.sun.star.container.XIndexReplace`. Describes the numbering levels.
`TextAutoGrowHeight`	`boolean` — If `true`, the height of the shape is automatically expanded or shrunk when text is added or removed from the shape.
`TextAutoGrowWidth`	`boolean` — If `true`, the width of the shape is automatically expanded or shrunk when text is added or removed from the shape.
`TextContourFrame`	`boolean` — If `true`, the left edge of every line of text is aligned with the left edge of this shape.
`TextFitToSize`	`com.sun.star.drawing.TextFitToSizeType`. Determines how the text inside of the Shape is stretched to fit in the Shape. Possible values are NONE, PROPORTIONAL, ALLLINES, and RESIZEATTR.
`TextHorizontalAdjust`	`com.sun.star.drawing.TextHorizontalAdjust`. Adjusts the horizontal position of the text inside of the shape.
`TextVerticalAdjust`	`com.sun.star.drawing.TextVerticalAdjust`. Adjusts the vertical position of the text inside of the shape.
`TextLeftDistance`	`long` — Distance from the left edge of the shape to the left edge of the text.
`TextRightDistance`	`long` — Distance from the right edge of the shape to the right edge of the text.
`TextUpperDistance`	`long` — Distance from the upper edge of the shape to the upper edge of the text.
`TextLowerDistance`	`long` — Distance from the lower edge of the shape to the lower edge of the text.
`TextMaximumFrameHeight`	`long` — Maximum height of the surrounding frame.
`TextMaximumFrameWidth`	`long` — Maximum width of the surrounding frame.
`TextMinimumFrameHeight`	`long` — Minimum height of the surrounding frame.
`TextMinimumFrameWidth`	`long` — Minimum width of the surrounding frame.
`TextAnimationAmount`	`short` — Number of pixels that the text is moved in each animation step.
`TextAnimationCount`	`short` — Defines how many times the text animation is repeated.
`TextAnimationDelay`	`short` — Delay between the animation steps in thousandths of a second.
`TextAnimationDirection`	`com.sun.star.drawing.TextAnimationDirection`. This enumeration defines the direction that the text moves.
`TextAnimationKind`	`com.sun.star.drawing.TextAnimationKind`. Defines the type of animation.
`TextAnimationStartInside`	`boolean`. If true, the text is visible at the start of the animation.
`TextAnimationStopInside`	`boolean`. If true, the text is visible at the end of the animation.

Properties of com.sun.star.drawing.TextProperties	
TextWritingMode	com.sun.star.text.WritingMode. This value selects the writing mode for the text.

The service com.sun.star.drawing.TextProperties includes com.sun.star.style.ParagraphProperties and com.sun.star.style.CharacterProperties. Since these services contain optional properties, the properties actually supported by drawing shapes are listed. Refer to the API reference or explanations or *7.3.2 Text Documents - Working with Text Documents - Formatting*.

The service com.sun.star.drawing.TextProperties includes com.sun.star.style.ParagraphProperties and com.sun.star.style.CharacterProperties. Since these services contain many optional properties, we list the properties actually supported by drawing shapes. Please look up the explanations in the API reference or in *7.3.2 Text Documents - Working with Text Documents - Formatting*.

com.sun.star.style.CharacterProperties of drawing text	
CharAutoKerning	boolean
CharColor	long
CharContoured	boolean
CharCrossedOut	boolean
CharEmphasis	short
CharEscapement	short
CharEscapementHeight	byte
CharFontCharSet	short
CharFontFamily	short
CharFontName	string
CharFontPitch	short
CharFontStyleName	string
CharHeight	float
CharKerning	short
CharLocale	com.sun.star.lang.Locale
CharPosture	com.sun.star.awt.FontSlant
CharRelief	short
CharScaleWidth	short
CharShadowed	boolean
CharStrikeout	short
CharUnderline	short
CharUnderlineColor	long
CharUnderlineHasColor	boolean
CharWeight	float
CharWordMode	boolean

There are Asian counterparts for a number of character properties.

com.sun.star.style.CharacterPropertiesAsian of drawing shapes	
CharFontPitchAsian	short
CharFontStyleNameAsian	string

com.sun.star.style.CharacterPropertiesAsian of drawing shapes	
CharHeightAsian	float
CharPostureAsian	com.sun.star.awt.FontSlant
CharLocaleAsian	com.sun.star.lang.Locale
CharWeightAsian	float

There is also a Complex flavor of the same properties:

com.sun.star.style.CharacterPropertiesComplex of drawing text	
CharFontPitchComplex	short
CharFontStyleNameComplex	string
CharHeightComplex	float
CharLocaleComplex	com.sun.star.lang.Locale
CharPostureComplex	com.sun.star.awt.FontSlant
CharWeightComplex	float

Paragraphs in drawing text support a selection of com.sun.star.style.ParagraphProperties:

Properties of com.sun.star.style.ParagraphProperties	
ParaAdjust	short
ParaBottomMargin	long
ParaFirstLineIndent	long
ParaIsCharacterDistance	boolean
ParaIsForbiddenRules	boolean
ParaIsHangingPunctuation	boolean
ParaIsHyphenation	boolean
ParaLastLineAdjust	short
ParaLeftMargin	long
ParaLineSpacing	com.sun.star.style.LineSpacing
ParaRightMargin	long
ParaTabStops	sequence <com.sun.star.style.TabStop >
ParaTopMargin	long
ParaUserDefinedAttributes	com.sun.star.uno.XInterface

The next example introduces a method that appends single text portions to a shape. It returns the XPropertySet interface of the text range that has been added. (Drawing/ShapeHelper.java)

```
/** add text to a shape.
    the return value is the PropertySet of the text range that has been added
 */
public static XPropertySet addPortion(XShape xShape, String sText, boolean bNewParagraph)
        throws com.sun.star.lang.IllegalArgumentException {
    XText xText = (XText)UnoRuntime.queryInterface(XText.class, xShape);

    XTextCursor xTextCursor = xText.createTextCursor();
    xTextCursor.gotoEnd(false);
    if (bNewParagraph) {
        xText.insertControlCharacter(xTextCursor, ControlCharacter.PARAGRAPH_BREAK, false);
        xTextCursor.gotoEnd(false);
    }
    XTextRange xTextRange = (XTextRange)UnoRuntime.queryInterface(XTextRange.class, xTextCursor);
    xTextRange.setString(sText);
    xTextCursor.gotoEnd(true);
    XPropertySet xPropSet = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, xTextRange);
    return xPropSet;
}
```

Using the previous method, the next example creates a rectangle shape that has a border of 2.5 cm with the text of two paragraphs is stretched by using the `com.sun.star.drawing.TextFitToSizeType` property. The text of the first paragraph is then colored green, and the second red. The *7.3.1 Text Documents - Working with Text Documents - Word Processing - Editing Text* provides further details of handling text. (Drawing/TextDemo.java)

```
createShape(xComponent, new Point(0,0),
    new Size(21000, 12500), "com.sun.star.drawing.RectangleShape");
xShapes.add(xRectangle);
xShapePropSet = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, xRectangle);

// TextFitToSize
xShapePropSet.setPropertyValue("TextFitToSize", TextFitToSizeType.PROPORTIONAL);

// border size
xShapePropSet.setPropertyValue("TextLeftDistance",  new Integer(2500));
xShapePropSet.setPropertyValue("TextRightDistance", new Integer(2500));
xShapePropSet.setPropertyValue("TextUpperDistance", new Integer(2500));
xShapePropSet.setPropertyValue("TextLowerDistance", new Integer(2500));

xTextPropSet = ShapeHelper.addPortion(xRectangle, "using TextFitToSize", false);
xTextPropSet.setPropertyValue("ParaAdjust", ParagraphAdjust.CENTER);
xTextPropSet.setPropertyValue("CharColor",  new Integer(0xff00 ));
xTextPropSet = ShapeHelper.addPortion(xRectangle, "and a Border distance of 2,5 cm", true);
xTextPropSet.setPropertyValue("CharColor",  new Integer(0xff0000));
```

Many shapes cast shadows. The `ShadowProperties` controls how this shadow looks:

Properties of `com.sun.star.drawing.ShadowProperties`	
`Shadow`	`boolean` — Enables or disables the shadow of a shape.
`ShadowColor`	`long` — Color of the shadow of the shape.
`ShadowTransparence`	`short` — Defines the degree of transparency of the shadow in percent.
`ShadowXDistance`	`long` — Horizontal distance between the left edge of the shape and the shadow.
`ShadowYDistance`	`long` — Vertical distance between the top edge of the shape and the shadow.

Glue Points and Connectors

By default, there are four glue points available that are used within the properties `StartGlue-PointIndex` and `EndGluePointIndex`. If a connector connects to the top, bottom, left or right of a shape, a new glue point is not created. The four directions are declared in the following example.

The first example demonstrates how to create a `com.sun.star.drawing.ConnectorShape` and connect it to two other shapes using the glue point index property. (Drawing/GluePointDemo.java)

```
XDrawPagesSupplier xDrawPagesSupplier = (XDrawPagesSupplier)UnoRuntime.queryInterface(
    XDrawPagesSupplier.class, xComponent);
XDrawPages xDrawPages = xDrawPagesSupplier.getDrawPages();
XPage xDrawPages.getByIndex( 0 );

XShapes xShapes = (XShapes) UnoRuntime.queryInterface(XShapes.class, xPage);

// create two rectangles
XShape xShape1 = ShapeHelper.createShape(xDrawDoc, new Point(15000, 1000), new Size(5000, 5000),
    "com.sun.star.drawing.RectangleShape");

XShape xShape2 = ShapeHelper.createShape(xDrawDoc, new Point(2000, 15000), new Size(5000, 5000),
    "com.sun.star.drawing.EllipseShape");

// and a connector
XShape xConnector = ShapeHelper.createShape(xDrawDoc,
    new Point(0, 0), new Size(0, 0), "com.sun.star.drawing.ConnectorShape");

xShapes.add(xShape1);
xShapes.add(xShape2);
xShapes.add(xConnector);

XPropertySet xConnectorPropSet = (XPropertySet)UnoRuntime.queryInterface(
    XPropertySet.class, xConnector);

// Index value of 0 : the shape is connected at the top
```

```
// Index value of 1 : the shape is connected at the left
// Index value of 2 : the shape is connected at the bottom
// Index value of 3 : the shape is connected at the right

int nStartIndex = 3;
int nEndIndex   = 1;

// the "StartPosition" or "EndPosition" property needs not to be set
// if there is a shape to connect
xConnectorPropSet.setPropertyValue("StartShape", xShape1);
xConnectorPropSet.setPropertyValue("StartGluePointIndex", new Integer(nStartIndex));

xConnectorPropSet.setPropertyValue("EndShape", xShape2);
xConnectorPropSet.setPropertyValue("EndGluePointIndex", new Integer(nEndIndex));
```

The next example demonstrates the usage of user defined glue points. (Drawing/GluePointDemo.java)

```
XGluePointsSupplier  xGluePointsSupplier;
XIndexContainer      xIndexContainer;
XIdentifierContainer xIdentifierContainer;

/* take care to use the structure GluePoint2 and not
   GluePoint, because otherwise the XIdentifierContainer
   won't accept it
*/
GluePoint2 aGluePoint = new GluePoint2();
aGluePoint.IsRelative = false;
aGluePoint.PositionAlignment = Alignment.CENTER;
aGluePoint.Escape = EscapeDirection.SMART;
aGluePoint.IsUserDefined = true;
aGluePoint.Position.X = 0;
aGluePoint.Position.Y = 0;

// create and insert a glue point at shape1
xGluePointsSupplier = (XGluePointsSupplier)UnoRuntime.queryInterface(
    XGluePointsSupplier.class, xShape1);
xIndexContainer = xGluePointsSupplier.getGluePoints();
xIdentifierContainer = (XIdentifierContainer)UnoRuntime.queryInterface(
    XIdentifierContainer.class, xIndexContainer);
int nIndexOfGluePoint1 = xIdentifierContainer.insert(aGluePoint);

// create and insert a glue point at shape2
xGluePointsSupplier = (XGluePointsSupplier)
    UnoRuntime.queryInterface(XGluePointsSupplier.class, xShape2);
xIndexContainer = xGluePointsSupplier.getGluePoints();
xIdentifierContainer = (XIdentifierContainer)UnoRuntime.queryInterface(
    XIdentifierContainer.class, xIndexContainer);
int nIndexOfGluePoint2 = xIdentifierContainer.insert(aGluePoint);

// create and add a connector
XShape xConnector2 = ShapeHelper.createShape(xDrawDoc,
    new Point(0, 0), new Size(0, 0), "com.sun.star.drawing.ConnectorShape");
xShapes.add( xConnector2 );

XPropertySet xConnector2PropSet = (XPropertySet)UnoRuntime.queryInterface(
    XPropertySet.class, xConnector2);

xConnector2PropSet.setPropertyValue("StartShape", xShape1);
xConnector2PropSet.setPropertyValue("StartGluePointIndex", new Integer(nIndexOfGluePoint1));

xConnector2PropSet.setPropertyValue( "EndShape", xShape2 );
xConnector2PropSet.setPropertyValue( "EndGluePointIndex", new Integer(nIndexOfGluePoint2));
```

Layer Handling

In Draw and Impress, each shape is associated to exactly *one* layer. The layer has properties that specify if connected shapes are visible, printable or editable.

The service `com.sun.star.drawing.DrawingDocument` implements the interface `com.sun.star.drawing.XLayerSupplier` that gives access to the `com.sun.star.drawing.XLayerManager` interface. The `com.sun.star.drawing.XLayerManager` interface is used to create and edit a layer, and is used to attach a layer to a shape. (Drawing/LayerDemo.java)

```
XShapes xShapes = (XShapes)UnoRuntime.queryInterface(XShapes.class, xPage);

XShape xRect1 = ShapeHelper.createShape(xComponent, new Point(1000, 1000), new Size(5000, 5000),
    "com.sun.star.drawing.RectangleShape");
```

```
    XShape xRect2 = ShapeHelper.createShape(xComponent, new Point(1000, 7000), new Size(5000, 5000),
        "com.sun.star.drawing.RectangleShape" );

    xShapes.add(xRect1);
    xShapes.add(xRect2);
    XPropertySet xTextProp = ShapeHelper.addPortion(xRect2, "this shape is locked", false);
    xTextProp.setPropertyValue("ParaAdjust", ParagraphAdjust.CENTER);
    ShapeHelper.addPortion(xRect2, "and the shape above is not visible", true);
    ShapeHelper.addPortion(xRect2, "(switch to the layer view to gain access)", true);

    // query for the XLayerManager
    XLayerSupplier xLayerSupplier = (XLayerSupplier)UnoRuntime.queryInterface(
        XLayerSupplier.class, xComponent);
    XNameAccess xNameAccess = xLayerSupplier.getLayerManager();
    XLayerManager xLayerManager = (XLayerManager)UnoRuntime.queryInterface(
        XLayerManager.class, xNameAccess);

    // create a layer and set its properties
    XPropertySet xLayerPropSet;
    XLayer xNotVisibleAndEditable = xLayerManager.insertNewByIndex(xLayerManager.getCount());
    xLayerPropSet = (XPropertySet)UnoRuntime.queryInterface(
        XPropertySet.class, xNotVisibleAndEditable);
    xLayerPropSet.setPropertyValue("Name", "NotVisibleAndEditable");
    xLayerPropSet.setPropertyValue("IsVisible", new Boolean(false));
    xLayerPropSet.setPropertyValue("IsLocked", new Boolean(true));

    // create a second layer
    XLayer xNotEditable = xLayerManager.insertNewByIndex(xLayerManager.getCount());
    xLayerPropSet = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, xNotEditable);
    xLayerPropSet.setPropertyValue("Name", "NotEditable" );
    xLayerPropSet.setPropertyValue("IsVisible", new Boolean(true));
    xLayerPropSet.setPropertyValue("IsLocked", new Boolean(true));

    // attach the layer to the rectangles
    xLayerManager.attachShapeToLayer(xRect1, xNotVisibleAndEditable);
    xLayerManager.attachShapeToLayer(xRect2, xNotEditable);
```

9.3.3 Inserting Files

Currently it is not possible to insert slides from a drawing or presentation into a drawing document through API. To accomplish this, use the **Insert – File** command from the menu.

9.3.4 Navigating

Initially, shapes in a document can only be accessed by their index. The only method to get more information about a shape on the page is to test for the shape type, so it is impossible to identify a particular shape. However, after a shape is inserted, you can name it in the user interface or through the shape interface `com.sun.star.container.XNamed`, and identify the shape by its name after retrieving it by index. Shapes cannot be *accessed* by their names.

Searching and replacing text in Drawing documents retrieves the shapes that contain the text that is searched for. For more information, refer to *6.2.9 Office Development - Common Application Features - Search and Replace*.

9.4 Handling Presentation Document Files

9.4.1 Creating and Loading Presentation Documents

The URL that must be used with `loadComponentFromURL()` for new presentation documents is "`private:factory/simpress`".

To avoid the initial dialog in new presentation documents, set the property `Silent` defined in `com.sun.star.document.MediaDescriptor` to true. This property has to be used with the sequence of `PropertyValue` structs that is passed to `loadComponentFromURL()`.

The snippet below loads a new presentation document in silent mode:

```
// the method getRemoteServiceManager is described in the chapter First Steps
mxRemoteServiceManager = this.getRemoteServiceManager();

// retrieve the Desktop object, we need its XComponentLoader
Object desktop = mxRemoteServiceManager.createInstanceWithContext(
    "com.sun.star.frame.Desktop", mxRemoteContext);

// query the XComponentLoader interface from the Desktop service
XComponentLoader xComponentLoader = (XComponentLoader)UnoRuntime.queryInterface(
    XComponentLoader.class, desktop);

// define load properties according to com.sun.star.document.MediaDescriptor
// the boolean property Silent tells the office to suppress the impress startup wizard
PropertyValue[] loadProps = new PropertyValue[1];
loadProps[0] = new PropertyValue();
loadProps[0].Name = "Silent";
loadProps[0].Value = new Boolean(true);

// load
com.sun.star.uno.XComponent xComponentLoader.loadComponentFromURL(
    "private:factory/simpress", "_blank", 0, loadProps);
```

9.4.2 Printing Presentation Documents

Presentation documents have the following specific properties to define if the notes and outline view should be printed:

Properties of `com.sun.star.presentation.DocumentSettings`	
`IsPrintNotes`	`boolean` — Specifies if the notes are also printed.
`IsPrintOutline`	`boolean` — Specifies if the outline is also printed.

9.6.2 Drawing - Overall Document Features - Settings describes how these settings are used.

9.5 Working with Presentation Documents

9.5.1 Presentation Document

The structure of Impress documents is enhanced by a handout page per document, one notes page per draw page, and one notes master page for each master page. This means that the creation of normal draw and draw master pages automatically create corresponding notes and notes master pages. Due to this fact there are no interfaces for creation or deletion of notes or notes master pages.

The following UML diagram describes the whole page structure of Impress. The horizontal dotted line illustrates the general page structure lying beneath the dotted line, and the enhanced page structure of Impress lying above.

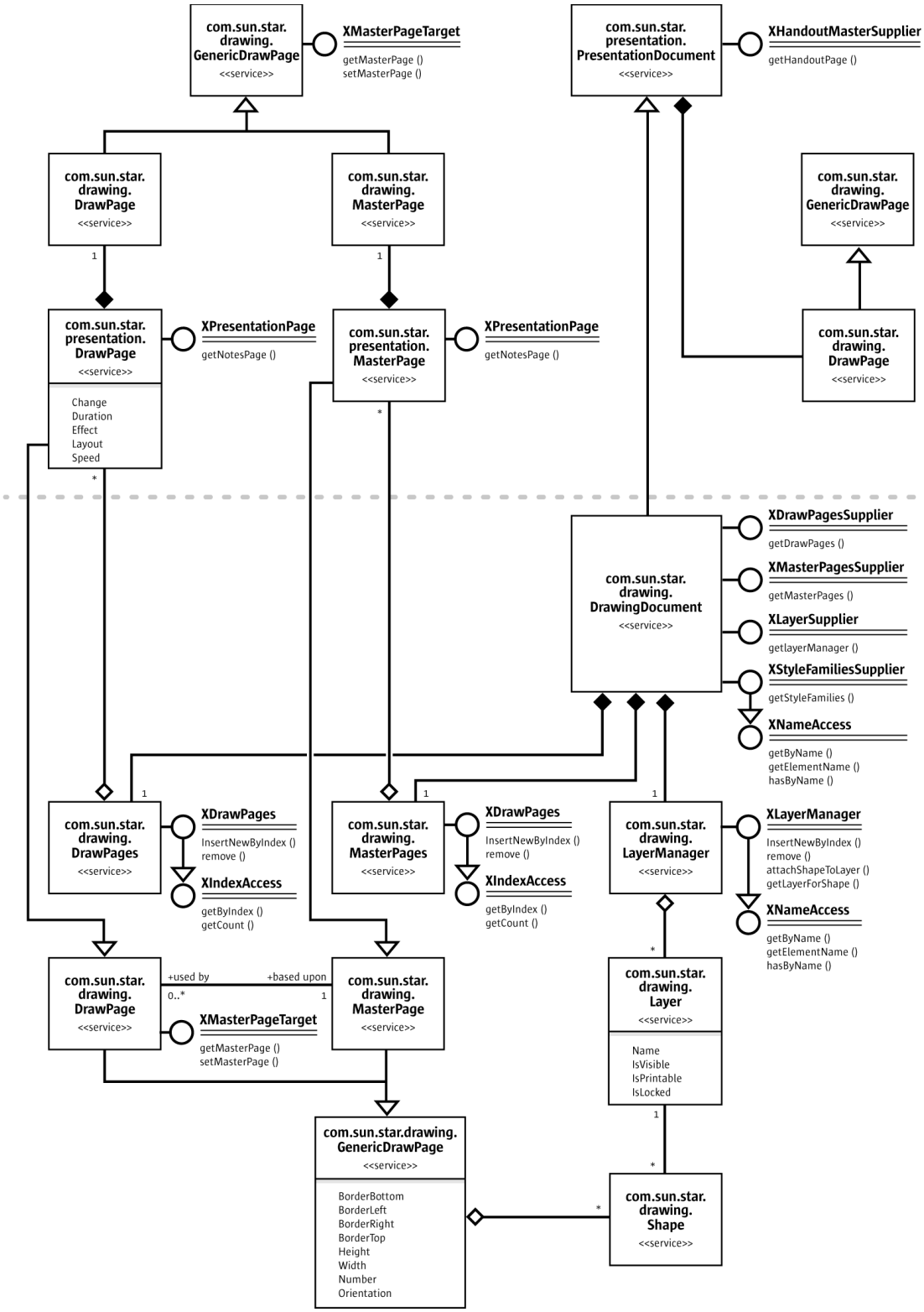


Illustration 138: PresentationDocument

Calling `getDrawPages()` at the `com.sun.star.drawing.XDrawPagesSupplier` interface of a presentation document retrieves a collection of `com.sun.star.presentation.DrawPage` instances with presentation specific properties.

The following two examples demonstrate how to access the notes pages and the handout page of an Impress document: (Drawing/PageHelper.java)

```
/** in Impress documents each draw page as also each draw master page has
    a corresponding notes page
*/
static public XDrawPage getNotesPage(XDrawPage xDrawPage) {
    XDrawPage xNotesPage;

    XPresentationPage xPresentationPage = (XPresentationPage)UnoRuntime.queryInterface(
        XPresentationPage.class, xDrawPage);

    /* only Impress pages support the XPresentationPage interface,
       for all other pages the interface will be zero, so a test
       won't hurt
    */
    if (xPresentationPage != null)
        xNotesPage = xPresentationPage.getNotesPage();

    return xNotesPage;
}
```

The notes master page that corresponds to a notes page can be accessed by the `com.sun.star.presentation.XPresentation` interface of the master page. (Drawing/PageHelper.java)

```
/** in impress each document has one handout page */
static public XDrawPage getHandoutMasterPage(XComponent xComponent) {
    XHandoutMasterSupplier aHandoutMasterSupplier =
        (XHandoutMasterSupplier)UnoRuntime.queryInterface(
             XHandoutMasterSupplier.class, xComponent);

    return aHandoutMasterSupplier.getHandoutMasterPage();
}
```

9.5.2 Presentation Settings

Impress documents contain a `Presentation` service that controls a running presentation. This `com.sun.star.presentation.Presentation` service can be accessed through the `com.sun.star.presentation.XPresentationSupplier` interface through the method:

```
com::sun::star::presentation::XPresentation getPresentation()
```

The method `getPresentation()` returns a `com.sun.star.presentation.Presentation` service. It contains properties for presentation settings and the interface `com.sun.star.presentation.XPresentation`.

The presentation settings define the slide range, which custom show is used, and how the presentation is executed. These settings are provided as properties of the service `com.sun.star.presentation.Presentation`. This service also exports the [IDL.com.sun.star.presentation.XPresentation] interface that starts and ends a presentation.

Methods of `com.sun.star.presentation.XPresentation`	
`start()`	Starts the presentation in full-screen mode.
`end()`	Stops the presentation.
`rehearseTimings()`	Starts the presentation from the beginning and shows the actual running time to the user.

Properties of `com.sun.star.presentation.Presentation`	
`AllowAnimations`	`boolean` — Enables/disables the shape animations.
`CustomShow`	`string` — Contains the name of a customized show that is used for the presentation.

Properties of com.sun.star.presentation.Presentation	
FirstPage	string — Contains the name of the page where the presentation is started.
IsAlwaysOnTop	boolean — If true, the window of the presentation is always on top of all the other windows.
IsAutomatic	boolean — If true, all pages are changed automatically.
IsEndless	boolean — If true, the presentation is repeated endlessly.
IsFullScreen	boolean — If true, the presentation runs in full-screen mode.
IsLivePresentation	boolean — With this property, the presentation is set to live mode.
IsMouseVisible	boolean — If true, the mouse is visible during the presentation.
Pause	long — Duration of the black screen after the presentation has finished.
StartWithNavigator	boolean — If true, the Navigator is opened at the start of the presentation.
UsePen	boolean — If true, a pen is shown during presentation.
IsShowAll	boolean — Show all slides.
IsShowLogo	boolean — Show OpenOffice.org logo on pause page in automatic mode.
IsTransitionOn-Click	boolean — Slide change on mouse click, in addition to pressing cursor right.

The properties IsShowAll, IsShowLogo and IsTransitionOnClick are currently not documented in the API reference.

The next example demonstrates how to start a presentation that is automatically repeated and plays in full-screen mode by modifying the presentation settings.
(Drawing/PresentationDemo.java)

```
XPresentationSupplier xPresSupplier = (XPresentationSupplier)UnoRuntime.queryInterface(
    XPresentationSupplier.class, xComponent);
XPresentation xPresentation = xPresSupplier.getPresentation();
XPropertySet xPresPropSet = (XPropertySet)UnoRuntime.queryInterface(
    XPropertySet.class, xPresentation);
xPresPropSet.setPropertyValue("IsEndless", new Boolean(true));
xPresPropSet.setPropertyValue("IsFullScreen", new Boolean(true));
xPresPropSet.setPropertyValue("Pause", new Integer(0));
xPresentation.start();
```

Custom Slide Show

Custom presentations are available at the com.sun.star.presentation.XCustomPresentationSupplier interface of the presentation document. It contains the method:

```
com::sun::star::container::XNameContainer getCustomPresentations()
```

The method getCustomPresentations() returns a com.sun.star.presentation.CustomPresentationAccess service that consists of the interfaces com.sun.star.container.XNameContainer and com.sun.star.lang.XSingleServiceFactory. The standard API interface com.sun.star.container.XNameContainer derived from com.sun.star.container.XNameReplace obtains existing Custom Presentations *and* to add new custom presentations by name. It introduces the methods:

```
void replaceByName( [in] string aName, [in] any aElement)

void insertByName( [in] string aName, [in] any aElement)
void removeByName( [in] string Name)
```

To add a new CustomPresentation, create it using createInstance() at the XSingleServiceFactory interface of the CustomPresentationAccess.

Methods of com.sun.star.lang.XSingleServiceFactory:

```
com::sun::star::uno::XInterface createInstance()
com::sun::star::uno::XInterface createInstanceWithArguments( [in] sequence< any aArguments >)
```

The CustomPresentation is now created. Its content consists of a com.sun.star.presentation.CustomPresentation. From the API, it is a named container of selected presentation draw pages. Draw pages can be added to a custom presentation or removed using its interface com.sun.star.container.XIndexContainer. In addition to the methods of an XIndexAccess, this standard API interface supports the following operations:

Methods introduced by com.sun.star.container.XIndexContainer:

```
void replaceByIndex( [in] long Index, [in] any Element)
void insertByIndex( [in] long Index, [in] any Element)
void removeByIndex( [in] long Index)
```

The name of a CustomPresentation is read and written using the interface com.sun.star.container.XNamed:

Methods of XNamed:

```
string getName()
void setName( [in] string aName)
```

A custom show is a collection of slides in a user-defined order that can be executed as a presentation. It is also possible to use a slide twice or skip slides. For instance, it is possible to create a short version of a presentation and a long version within the same document. The number of custom shows is unlimited.

The next example demonstrates how to create two custom shows and set one of them as an active presentation. (Drawing/CustomShowDemo.java)

```
XDrawPagesSupplier xDrawPagesSupplier = (XDrawPagesSupplier)UnoRuntime.queryInterface(
    XDrawPagesSupplier.class, xComponent);
XDrawPages xDrawPages = xDrawPagesSupplier.getDrawPages();

// take care that this document has ten pages
while (xDrawPages.getCount() < 10)
    xDrawPages.insertNewByIndex(0);

// assign a name to each page
String aNameArray[] = {"Introduction", "page one", "page two", "page three", "page four",
                       "page five", "page six", "page seven", "page eight", "page nine"};
int i;
for (i = 0; i < 10; i++) {
    XNamed xPageName = (XNamed)UnoRuntime.queryInterface(XNamed.class, xDrawPages.getByIndex(i));
    xPageName.setName(aNameArray[i]);
}

/* create two custom shows, one will play slide 6 to 10 and is named "ShortVersion"
   the other one will play slide 2 til 10 and is named "LongVersion"
*/
XCustomPresentationSupplier xCustPresSupplier = (XCustomPresentationSupplier)
    UnoRuntime.queryInterface(XCustomPresentationSupplier.class, xComponent);

/* the following container is a container for further container
   which concludes the list of pages that are to play within a custom show
*/
XNameContainer xNameContainer = xCustPresSupplier.getCustomPresentations();
XSingleServiceFactory xFactory = (XSingleServiceFactory)UnoRuntime.queryInterface(
    XSingleServiceFactory.class, xNameContainer);

Object xObj;
XIndexContainer xContainer;

/* instanciate an IndexContainer that will take
   a list of draw pages for the first custom show
*/
xObj = xFactory.createInstance();
xContainer = (XIndexContainer)
    UnoRuntime.queryInterface(XIndexContainer.class, xObj);
for (i = 5; i < 10; i++)
    xContainer.insertByIndex(xContainer.getCount(), xDrawPages.getByIndex(i));
```

```
xNameContainer.insertByName("ShortVersion", xContainer);

/* instanciate an IndexContainer that will take
   a list of draw page for a second custom show
*/
xObj = xFactory.createInstance();
xContainer = (XindexContainer)UnoRuntime.queryInterface(XIndexContainer.class, xObj);
for (i = 1; i < 10; i++)
    xContainer.insertByIndex(xContainer.getCount(), xDrawPages.getByIndex(i));
xNameContainer.insertByName("LongVersion", xContainer);

/* which custom show is to use
   can been set in the presentation settings
*/

XPresentationSupplier xPresSupplier = (XPresentationSupplier)UnoRuntime.queryInterface(
    XPresentationSupplier.class, xComponent);
XPresentation xPresentation = xPresSupplier.getPresentation();
XPropertySet xPresPropSet = (XPropertySet)UnoRuntime.queryInterface(
    XPropertySet.class, xPresentation);
xPresPropSet.setPropertyValue("CustomShow", "ShortVersion");
```

Presentation Effects

There are two kinds of presentation effects, the *fading* of one page to another, and the *animation* of objects and texts within a slideshow.

Slide Transition

In Impress, each page has its own slide transition that can be composed by the properties of the service `com.sun.star.presentation.DrawPage`.

Setting the following properties enables slide transition:

Properties of `com.sun.star.presentation.DrawPage`	
`Change`	`long` — Specifies how the page change is triggered. If this is 0, the user must click to start each object animation and to change the page. If set to 1, the page is automatically switched. If it is set to 2, all object effects run automatically, but the user has to click on the page to change it.
`Duration`	`long` — If the property `Change` is set to 1, this property is the time in seconds the page is shown, before switching to the next page.
`Effect`	`com.sun.star.presentation.FadeEffect`. This is the effect that is used to fade in the page.
`Speed`	`com.sun.star.presentation.AnimationSpeed`. Defines the speed of the fade-in effect of the page. Possible values are: • `SLOW` sets the speed from the animation or fade to slow. • `MEDIUM` sets the speed from the animation or fade to medium. • `FAST` sets the speed from the animation or fade to fast.
`Layout`	`short` — This number specifies a presentation layout for this page, if this property is not `ZERO`.

The next table contains all available `com.sun.star.presentation.FadeEffect` enum values:

`NONE`	`RANDOM`	`DISSOLVE`

FADE_FROM_LEFT FADE_FROM_RIGHT FADE_FROM_TOP FADE_FROM_BOTTOM FADE_FROM_UPPERLEFT FADE_FROM_UPPERRIGHT FADE_FROM_LOWERLEFT FADE_FROM_LOWERRIGHT	MOVE_FROM_LEFT MOVE_FROM_RIGHT MOVE_FROM_TOP MOVE_FROM_BOTTOM MOVE_FROM_UPPERLEFT MOVE_FROM_UPPERRIGHT MOVE_FROM_LOWERRIGHT MOVE_FROM_LOWERLEFT	UNCOVER_TO_LEFT UNCOVER_TO_RIGHT UNCOVER_TO_TOP UNCOVER_TO_BOTTOM UNCOVER_TO_UPPERLEFT UNCOVER_TO_UPPERRIGHT UNCOVER_TO_LOWERRIGHT UNCOVER_TO_LOWERLEFT
FADE_TO_CENTER FADE_FROM_CENTER	VERTICAL_STRIPES HORIZONTAL_STRIPES	CLOCKWISE COUNTERCLOCKWISE
ROLL_FROM_LEFT ROLL_FROM_RIGHT ROLL_FROM_TOP ROLL_FROM_BOTTOM	CLOSE_VERTICAL CLOSE_HORIZONTAL OPEN_VERTICAL OPEN_HORIZONTAL	SPIRALIN_LEFT SPIRALIN_RIGHT SPIRALOUT_LEFT SPIRALOUT_RIGHT
WAVYLINE_FROM_LEFT WAVYLINE_FROM_RIGHT WAVYLINE_FROM_TOP WAVYLINE_FROM_BOTTOM	STRETCH_FROM_LEFT STRETCH_FROM_RIGHT STRETCH_FROM_TOP STRETCH_FROM_BOTTOM	VERTICAL_LINES HORIZONTAL_LINES
VERTICAL_CHECKERBOARD HORIZONTAL_CHECKERBOARD		

The following Java example shows how to set slide transition effects that are applied to the first page. (Drawing/PresentationDemo.java)

```
// set the slide transition effect of the first page
XDrawPagesSupplier xDrawPagesSupplier =(XDrawPagesSupplier)UnoRuntime.queryInterface(
    XDrawPagesSupplier.class, xComponent);

XDrawPages xDrawPages = xDrawPagesSupplier.getDrawPages();
XDrawPage  xDrawPage = (XDrawPage)xDrawPages.getByIndex(0);

xShapes = (XShapes)UnoRuntime.queryInterface(XShapes.class, xDrawPage);

XPropertySet xPropSet = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, xDrawPage);

// set the slide transition effect properties
xPropSet.setPropertyValue("Effect", com.sun.star.presentation.FadeEffect.RANDOM);
xPropSet.setPropertyValue("Speed", com.sun.star.presentation.AnimationSpeed.MEDIUM);

/* Change specifies how the page change is triggered. If this
   is 0, the user must click to start each object animation
   and to change the page. If set to 1, the page is
   automatically switched. If it is set to 2, all object
   effects run automatically, but the user has to click on the
   page to change it.
*/
xPropSet.setPropertyValue("Change", new Integer(1));

/* If the property DrawPage::Change is set to 1, Duration specifies the
   time in seconds the page is shown before switching to the next page.
*/
xPropSet.setPropertyValue("Duration", new Integer(5));
```

Animations and Interactions

In a Presentation, each shape of the draw and master page provides the `com.sun.star.presentation.Shape` service with a number of properties that describe the manner the shape is displayed or acting in a presentation.

There are two kinds of shape effects. The first kind of effects are visual changes, such as animations and dimming effects. The second kind of effects are `OnClick` actions. All of these effects are controlled by the properties of a presentation shape:

Properties of `com.sun.star.presentation.Shape`	
`OnClick`	`com.sun.star.presentation.ClickAction`. Selects an action performed after the user clicks on this shape. Possible values are: • `NONE` - no action is performed on click • `PREVPAGE` - the presentation jumps to the previous page • `NEXTPAGE` - the presentation jumps to the next page • `FIRSTPAGE` - the presentation continues with the first page • `LASTPAGE` - the presentation continues with the last page • `BOOKMARK` - the presentation jumps to the bookmark URL defined in the shape property `Bookmark` • `DOCUMENT` - the presentation jumps to the document given in `Bookmark`. It selects the object whose name is given after a # in the `Bookmark` URL. • `INVISIBLE` - the object renders itself invisible after a click • `SOUND` - the sound specified in `Sound` is played after a click • `VERB` - the OLE verb specified in the shape property `Verb` is performed on this object • `VANISH` - the object vanishes with the effect specified in the property Effect • `PROGRAM` - the program specified in `Bookmark` is executed after a click • `MACRO` - the OpenOffice.org Basic macro specified in `Bookmark` is executed after the click. For the syntax of Basic macro URLs, refer to the chapter *11 Basic and Dialogs*. • `STOPPRESENTATION` - the presentation is stopped after the click
`Bookmark`	`string` — A generic URL for the property `OnClick`.
`Verb`	`long` — Valid only for OLE shapes. Specifies an "OLE2" verb for the `ClickAction VERB` in the property `OnClick`. For possible verbs, select the OLE shape, and point the cursor to **Edit – Object**. The order of appearance corresponds to the value needed for `Verb`.
`DimPrevious`	`boolean` — Only valid when `Effect` contains an `AnimationEffect`. If true, this shape is painted using `DimColor` on the next click after finishing the `AnimationEffect`.
`DimHide`	`boolean` — Only valid when `Effect` contains an `AnimationEffect`. If this property and the property `DimPrevious` are both true, the shape is hidden on the next click after the `AnimationEffect` has finished.
`DimColor`	`long` — Only valid when `Effect` contains an `AnimationEffect`. This color is used to paint the shape on the next click after the animation effect has finished. The property `DimPrevious` must be true and `DimHide` must be false for this property to work.
`Effect`	`com.sun.star.presentation.AnimationEffect`. Selects the animation effect of this shape. For possible values see the table below.

Properties of com.sun.star.presentation.Shape	
PresentationOrder	long — This is the position of this shape in the order of the shapes that can be animated on its page. The animations are executed in the order given in PresentationOrder, starting at the shape with the PresentationOrder 1. You can change the order by changing this number. Setting it to 1 makes this shape the first shape in the execution order for the animation effects.
SoundOn	boolean — If true, the sound file specified in Sound is played while the animation effect is executed.
Sound	string — This is the URL to a sound file that is played while the animation effect of this shape is running.
PlayFull	boolean. — If true, the sound specified in the Sound property of this shape is played completely. If false, the sound stops after completing the AnimationEffect specified in Effect.
Speed	com.sun.star.presentation.AnimationSpeed. This is the speed of the animation effect. Possible values: SLOW, MEDIUM, and FAST.
TextEffect	com.sun.star.presentation.AnimationEffect. This is the animation effect for the text inside this shape. For possible values, see the table below.
IsEmptyPresentationObject	[readonly] boolean — If this is a default presentation object and if it is empty, this property is true.
IsPresentationObject	[readonly] boolean — If true, a shape is part of the current AutoLayout and is considered a *presentation object*. AutoLayouts are predefined page layouts consisting of shapes, such as a title box and an outline box.

The next table contains all available com.sun.star.presentation.AnimationEffect enums.

NONE	RANDOM	DISSOLVE
APPEAR	HIDE	PATH
FADE_FROM_LEFT FADE_FROM_RIGHT FADE_FROM_TOP FADE_FROM_BOTTOM FADE_FROM_UPPERLEFT FADE_FROM_UPPERRIGHT FADE_FROM_LOWERLEFT FADE_FROM_LOWERRIGHT	MOVE_FROM_LEFT MOVE_FROM_RIGHT MOVE_FROM_TOP MOVE_FROM_BOTTOM MOVE_FROM_UPPERLEFT MOVE_FROM_UPPERRIGHT MOVE_FROM_LOWERRIGHT MOVE_FROM_LOWERLEFT	ZOOM_IN_FROM_LEFT ZOOM_IN_FROM_RIGHT ZOOM_IN_FROM_TOP ZOOM_IN_FROM_BOTTOM ZOOM_IN_FROM_UPPERLEFT ZOOM_IN_FROM_UPPERRIGHT ZOOM_IN_FROM_LOWERRIGHT ZOOM_IN_FROM_LOWERLEFT
CLOCKWISE COUNTERCLOCKWISE	CLOSE_VERTICAL CLOSE_HORIZONTAL	OPEN_VERTICAL OPEN_HORIZONTAL
LASER_FROM_LEFT LASER_FROM_RIGHT LASER_FROM_TOP LASER_FROM_BOTTOM LASER_FROM_UPPERLEFT LASER_FROM_UPPERRIGHT LASER_FROM_LOWERLEFT LASER_FROM_LOWERRIGHT	MOVE_TO_LEFT MOVE_TO_RIGHT MOVE_TO_TOP MOVE_TO_BOTTOM MOVE_TO_UPPERLEFT MOVE_TO_UPPERRIGHT MOVE_TO_LOWERRIGHT MOVE_TO_LOWERLEFT	MOVE_SHORT_TO_LEFT MOVE_SHORT_TO_RIGHT MOVE_SHORT_TO_TOP MOVE_SHORT_TO_BOTTOM MOVE_SHORT_TO_UPPERLEFT MOVE_SHORT_TO_UPPERRIGHT MOVE_SHORT_TO_LOWERRIGHT MOVE_SHORT_TO_LOWERLEFT

ZOOM_OUT_FROM_LEFT ZOOM_OUT_FROM_RIGHT ZOOM_OUT_FROM_TOP ZOOM_OUT_FROM_BOTTOM ZOOM_OUT_FROM_UPPERLEFT ZOOM_OUT_FROM_UPPERRIGHT ZOOM_OUT_FROM_LOWERRIGHT ZOOM_OUT_FROM_LOWERLEFT	STRETCH_FROM_LEFT STRETCH_FROM_RIGHT STRETCH_FROM_TOP STRETCH_FROM_BOTTOM STRETCH_FROM_UPPERLEFT STRETCH_FROM_UPPERRIGHT STRETCH_FROM_LOWERRIGHT STRETCH_FROM_LOWERLEFT	MOVE_SHORT_FROM_LEFT MOVE_SHORT_FROM_RIGHT MOVE_SHORT_FROM_TOP MOVE_SHORT_FROM_BOTTOM MOVE_SHORT_FROM_UPPERLEFT MOVE_SHORT_FROM_UPPERRIGHT MOVE_SHORT_FROM_LOWERRIGHT MOVE_SHORT_FROM_LOWERLEFT
WAVYLINE_FROM_LEFT WAVYLINE_FROM_RIGHT WAVYLINE_FROM_TOP WAVYLINE_FROM_BOTTOM	SPIRALIN_LEFT SPIRALIN_RIGHT SPIRALOUT_LEFT SPIRALOUT_RIGHT	FADE_FROM_CENTER FADE_TO_CENTER VERTICAL_STRIPES HORIZONTAL_STRIPES
ZOOM_IN ZOOM_IN_SMALL ZOOM_IN_SPIRAL	ZOOM_OUT ZOOM_OUT_SMALL ZOOM_OUT_SPIRAL	VERTICAL_LINES HORIZONTAL_LINES
ZOOM_IN_FROM_CENTER ZOOM_OUT_FROM_CENTER	VERTICAL_CHECKERBOARD HORIZONTAL_CHECKERBOARD	VERTICAL_ROTATE HORIZONTAL_ROTATE
HORIZONTAL_STRETCH VERTICAL_STRETCH		

The next example demonstrates how to set object effects and object interaction.

The example use a method `createAndInsertShape()` from the `ShapeHelper` class. It takes the drawing document, the `XShapes` interface of the `DrawPage` the shape is to be inserted in, the position and size of the new shape, and the service name of the required shape. It delegates shape creation to the helper method `createShape()` and inserts the new shape into the given `XShapes` container. Finally, it retrieves the `XPropertySet` interface of the inserted shape and returns it to the caller. (Drawing/ShapeHelper.java)

```
public static XPropertySet createAndInsertShape( XComponent xDrawDoc,
        XShapes xShapes, Point aPos, Size aSize, String sShapeType) throws java.lang.Exception {
    XShape xShape = createShape(xDrawDoc, aPos, aSize, sShapeType);
    xShapes.add(xShape);
    XPropertySet xPropSet = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, xShape);
    return xPropSet;
}
```

The following example shows animations and `OnClick` actions for four shapes. On click, the first shape builds up in a wavy line from the bottom and is dimmed (painted) red afterwards. The second shape is hidden on click. Clicking the third shape makes the presentation jump to the first page, whereas clicking the fourth shape jumps to a bookmark. The bookmark contains the name of the second slide "page – two". (Drawing/PresentationDemo.java)

```
    XShapes      xShapes;
    XPropertySet xShapePropSet;

    XDrawPagesSupplier xDrawPagesSupplier = (XDrawPagesSupplier)UnoRuntime.queryInterface(
        XDrawPagesSupplier.class, xComponent);
    XDrawPages xDrawPages = xDrawPagesSupplier.getDrawPages();

    // create pages, so that three are available
    while (xDrawPages.getCount() < 3)
        xDrawPages.insertNewByIndex(0);

    // get the shape container for page one
    xShapes = (XShapes)UnoRuntime.queryInterface(
        XShapes.class, (XDrawPage)xDrawPages.getByIndex(0));

    // create a rectangle that is placed on the top left of the page
    xShapePropSet = ShapeHelper.createAndInsertShape( xComponent,
        xShapes, new Point(1000, 1000), new Size(5000, 5000),
        "com.sun.star.drawing.RectangleShape" );

    // and now set an object animation
    xShapePropSet.setPropertyValue("Effect",
        com.sun.star.presentation.AnimationEffect.WAVYLINE_FROM_BOTTOM);

    /* the following three properties provoke that the shape is dimmed to red
       after the animation has been finished
```

```
    */
    xShapePropSet.setPropertyValue("DimHide", new Boolean(false));
    xShapePropSet.setPropertyValue("DimPrevious", new Boolean(true));
    xShapePropSet.setPropertyValue("DimColor", new Integer(0xff0000));

    // get the shape container for the second page
    xShapes = (XShapes)UnoRuntime.queryInterface(XShapes.class, (XDrawPage)xDrawPages.getByIndex(1));

    // create an ellipse that is placed on the bottom right of second page
    xShapePropSet = ShapeHelper.createAndInsertShape( xComponent, xShapes,
        new Point(21000, 15000), new Size(5000, 5000), "com.sun.star.drawing.EllipseShape");
    xShapePropSet.setPropertyValue("Effect", com.sun.star.presentation.AnimationEffect.HIDE);

    /* create two objects for the third page.
       clicking the first object lets the presentation jump
       to page one by using ClickAction.FIRSTPAGE,
       the second object lets the presentation jump to page two
       by using a ClickAction.BOOKMARK
    */
    xShapes = (XShapes)UnoRuntime.queryInterface(XShapes.class,
          (XDrawPage)xDrawPages.getByIndex(2));
    xShapePropSet = ShapeHelper.createAndInsertShape(xComponent, xShapes,
        new Point(1000, 8000), new Size(5000, 5000),
        "com.sun.star.drawing.EllipseShape" );
    xShapePropSet.setPropertyValue("Effect",
com.sun.star.presentation.AnimationEffect.FADE_FROM_BOTTOM );
    xShapePropSet.setPropertyValue("OnClick", com.sun.star.presentation.ClickAction.FIRSTPAGE);

    xShapePropSet = ShapeHelper.createAndInsertShape(xComponent, xShapes,
        new Point(22000, 8000), new Size(5000, 5000),
        "com.sun.star.drawing.RectangleShape");
    xShapePropSet.setPropertyValue("Effect",
com.sun.star.presentation.AnimationEffect.FADE_FROM_BOTTOM);
    xShapePropSet.setPropertyValue( "OnClick", com.sun.star.presentation.ClickAction.BOOKMARK);

    // set the name of page two, and use it with the bookmark action
    XNamed xPageName = (XNamed)UnoRuntime.queryInterface(XNamed.class,
        (XDrawPage)xDrawPages.getByIndex(1));
    xPageName.setName("page - two");
    xShapePropSet.setPropertyValue("Bookmark", xPageName.getName());
```

9.6 Overall Document Features

9.6.1 Styles

Graphics Styles

Graphics Styles are available in drawing and presentation documents, and they control the formatting of the drawing shapes in drawing or presentation slides. In contrast to styles in text documents, the style property of a shape is not a string, but a `com.sun.star.style.XStyle`. To work with an existing graphics style, get the styles container from the `com.sun.star.style.XStyleFamiliesSupplier` and use its `com.sun.star.container.XNameAccess` to retrieve the style family named "graphics". The programmatic names of the style families in graphics are:

GUI name	**Programmatic name**	**Remark**
Default	standard	The style Default (standard) is used for newly inserted filled rectangles, filled ellipses, lines, connectors, text boxes, and 3D objects.
Dimension Line	measure	Used for newly inserted dimension lines.
First line indent	textbodyindent	Apply manually.
Heading	headline	Apply manually.

GUI name	Programmatic name	Remark
Heading1	headline1	Apply manually.
Heading2	headline2	Apply manually.
Object with Arrow	objectwitharrow	Apply manually.
Object with shadow	objectwithshadow	Apply manually.
Object without fill	objectwithoutfill	Used for newly inserted rectangles and ellipses without filling.
Text	text	Newly inserted text boxes do not use this style. They use Default and remove the fill settings for Default.
Text body	textbody	Apply manually.
Text body justified	textbodyjustfied	Apply manually.
Title	title	Apply manually.
Title1	title1	Apply manually.
Title2	title2	Apply manually.

There are two methods to change an applied shape style:

- Retrieve the style from the style family “graphics” by its programmatic name, change the properties, and put back into the style family using `replaceByName()` at the style family's `com.sun.star.container.XNameContainer` interface.
- Apply an existing style object that is not applied to a shape by setting the shape's `style` property.

New styles can be created, as well. For this purpose, use `createInstance()` at the document factory of a drawing document and ask for a "com.sun.star.style.Style" service. Set the properties of the new style, as required. Append the new style to the style family "graphics" using `insertByName()` at its `XNameContainer` interface. Now use the Style property of existing shapes to put the new style to work.

You can either change a currently applied shape style by retrieving it from the style family "graphics" by its programmatic name, changing its properties and putting it back into the style family using `replaceByName()` at the style family's `com.sun.star.container.XNameContainer` interface. Or you can apply an existing, but currently unapplied style object to a shape by setting the shape's `Style` property accordingly.

You can create new styles as well. For this purpose, use `createInstance()` at the document factory of a drawing document and ask for a "com.sun.star.style.Style" service. Set the properties of the new style as needed. Afterwards append the new style to the style family "graphics" using `insertByName()` at its `XNameContainer` interface. Now you can use the Style property of existing shapes in order to put your new style to work.

Styles created by the document factory support the properties of the following services:

- `com.sun.star.drawing.FillProperties`
- `com.sun.star.drawing.LineProperties`
- `com.sun.star.drawing.ShadowProperties`
- `com.sun.star.drawing.ConnectorProperties`
- `com.sun.star.drawing.MeasureProperties`
- `com.sun.star.style.ParagraphProperties`

- com.sun.star.style.CharacterProperties
- com.sun.star.drawing.TextProperties

Presentation Styles

Presentation styles are only available in presentation documents and control the formatting of the following parts of a presentation:

- title text
- subtitle text
- outline text
- background
- background shapes
- notes text

The corresponding style family has the programmatic name "Default" and is available at the XStyleFamiliesSupplier of a presentation document.

GUI Name	Programmatic Name	Remark
Title	title	Style for text of new title presentation objects.
Subtitle	subtitle	Style that is used for the presentation object on pages with a “Title Slide” layout.
Background	background	Style for the page background.
Background objects	backgroundobjects	Style for shapes on the background.
notes	Notes	Style for notes text.
outline1	Outline 1	Style for outline level 1.
outline2	Outline 2	Style for outline level 2.
outline3	Outline 3	Style for outline level 3.
outline4	Outline 4	Style for outline level 4.
outline5	Outline 5	Style for outline level 5.
outline6	Outline 6	Style for outline level 6.
outline7	Outline 7	Style for outline level 7.
outline8	Outline 8	Style for outline level 8.
outline9	Outline 9	Style for outline level 9.

Existing presentation styles can only be altered. New styles can not be created and a different presentation style cannot be applied other than the current one. The following example works with presentation styles: (Drawing/StyleDemo.java).

You can only alter existing presentation styles. You cannot create new styles and you cannot apply a different presentation style other than the current one. The following example works with presentation styles: (Drawing/StyleDemo.java).

```
// The first part of this demo will set each "CharColor" Property
// that is available within the styles of the document to red. It
// will also print each family and style name to the standard output
```

```
XModel xModel = (XModel)UnoRuntime.queryInterface(XModel.class, xComponent);
com.sun.star.style.XStyleFamiliesSupplier xSFS = (com.sun.star.style.XStyleFamiliesSupplier)
        UnoRuntime.queryInterface(com.sun.star.style.XStyleFamiliesSupplier.class, xModel);
com.sun.star.container.XNameAccess xFamilies = xSFS.getStyleFamilies();

// the element should now contain at least two Styles. The first is
// "graphics" and the other one is the name of the Master page
String[] Families = xFamilies.getElementNames();
for (int i = 0; i < Families.length; i++) {
    // this is the family
    System.out.println("\n" + Families[i]);

    // and now all available styles
    Object aFamilyObj = xFamilies.getByName(Families[i]);
    com.sun.star.container.XNameAccess xStyles = (com.sun.star.container.XNameAccess)
                UnoRuntime.queryInterface(com.sun.star.container.XNameAccess.class, aFamilyObj);
    String[] Styles = xStyles.getElementNames();
    for (int j = 0; j < Styles.length; j++) {
        System.out.println( "   " + Styles[j]);
        Object aStyleObj = xStyles.getByName(Styles[j]);
        com.sun.star.style.XStyle xStyle = (com.sun.star.style.XStyle)
        UnoRuntime.queryInterface(com.sun.star.style.XStyle.class, aStyleObj);
        // now we have the XStyle Interface and the CharColor for all styles
        // is exemplary be set to red.
        XPropertySet xStylePropSet = (XPropertySet)
            UnoRuntime.queryInterface( XPropertySet.class, xStyle );
        XPropertySetInfo xStylePropSetInfo = xStylePropSet.getPropertySetInfo();
        if (xStylePropSetInfo.hasPropertyByName("CharColor")) {
            xStylePropSet.setPropertyValue("CharColor", new Integer(0xff0000));
        }
    }
}

/* now create a rectangle and apply the "title1" style of
   the "graphics" family
*/
Object obj = xFamilies.getByName("graphics");
com.sun.star.container.XNameAccess xStyles = (XNameAccess)
    UnoRuntime.queryInterface(com.sun.star.container.XNameAccess.class, obj);
obj = xStyles.getByName("title1");
com.sun.star.style.XStyle xTitle1Style = (com.sun.star.style.XStyle)UnoRuntime.queryInterface(
    com.sun.star.style.XStyle.class, obj);

XDrawPagesSupplier xDrawPagesSupplier = (XDrawPagesSupplier)UnoRuntime.queryInterface(
    XDrawPagesSupplier.class, xComponent);
XDrawPages xDrawPages = xDrawPagesSupplier.getDrawPages();
XDrawPage xDrawPage = (XDrawPage)xDrawPages.getByIndex(0);
XShapes xShapes = (XShapes)UnoRuntime.queryInterface(XShapes.class, xDrawPage);
XShape xShape = ShapeHelper.createShape(xComponent, new Point(0, 0),
    new Size(5000, 5000), "com.sun.star.drawing.RectangleShape");
xShapes.add(xShape);
XPropertySet xPropSet = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, xShape);
xPropSet.setPropertyValue("Style", xTitle1Style);
```

9.6.2 Settings

To use the global document settings, the document service factory must be asked for a `com.sun.star.document.Settings` service using the method `createInstance()` at its `com.sun.star.lang.XMultiServiceFactory` interface. This service supports `com.sun.star.beans.PropertySet` and acts upon the current document by setting its properties.

The services `com.sun.star.drawing.DocumentSettings` and `com.sun.star.presentation.DocumentSettings` provide the following properties additionally to the global document settings.

Properties of `com.sun.star.drawing.DocumentSettings`	
`MeasureUnit`	`short` — this is the default logical measure unit that is used for string formatings inside the document.
`ScaleNumerator`	`long` — is the numerator for the logical scale of the document.
`ScaleDenominator`	`long` — is the denominator for the logical scale of the document.

Properties of `com.sun.star.drawing.DocumentSettings`	
`IsPrintFitPage`	`boolean` — this property enables or disables the fitting of the page to the printable area during print, `true` enable and `false` disable.
`IsPrintTilePage`	`boolean` — If the property `IsPrintTilePage` is set to 1 and the paper size for printing is larger than the paper size of the printer than the content is tiled over multiple pages.
`PageNumberFormat`	`long` — is the number format used for page number fields.
`ParagraphSummation`	`boolean` — If the property `ParagraphSummation` is set to 1, the distance between two paragraphs is the sum of ParaTopMargin of the previous and ParaBottomMargin of the next paragraph. If 0, only the greater of the two is choosen.

Properties of `com.sun.star.presentation.DocumentSettings`	
`IsPrintDrawing`	`boolean` — this property enables or disables the printing of the drawing pages, `true` enable and `false` disable.
`IsPrintNotes`	`boolean` — this property enables or disables the printing of the notes pages, `true` enable and `false` disable.
`IsPrintHandout`	`boolean` — this property enables or disables the printing of the handout pages, `true` enable and `false` disable.
`IsPrintOutline`	`boolean` — this property enables or disables the printing of the outline pages, `true` enable and `false` disable.
`IsPrintHiddenPages`	`boolean` — this property enables or disables the printing of draw pages that are marked hidden, `true` enable and `false` disable.
`IsPrintFitPage`	`boolean` — this property enables or disables the fitting of the page to the printable area during print, `true` enable and `false` disable.
`IsPrintTilePage`	`boolean` — If this property `IsPrintTilePage` is set to 1 and the paper size for printing is larger than the paper size of the printer than the content is tiled over multiple pages.
`PageNumberFormat`	`long` — is the number format used for page number fields.
`ParagraphSummation`	`boolean` — If the property `ParagraphSummation` is set to 1, the distance between two paragraphs is the sum of ParaTopMargin of the previous and ParaBottomMargin of the next paragraph. If 0, only the greater of the two is choosen.

9.6.3 Page Formatting

As opposed to text and spreadsheet documents, page formatting in drawings and presentations is not done through styles. Rather, hard format the following properties:

Properties of com.sun.star.drawing.GenericDrawPage	
BookmarkURL	string
BorderBottom	long
BorderLeft	long
BorderRight	long
BorderTop	long
Height	long
Number	short
Orientation	com.sun.star.view.PaperOrientation
Width	long

9.7 Drawing and Presentation Document Controller

The controller is available at the XModel interface of the document model:

```
com::sun::star::frame::XController getCurrentController()
```

9.7.1 Setting the Current Page, Using the Selection

The controller is a com.sun.star.drawing.DrawingDocumentDrawView that supports the following interfaces:

- com.sun.star.drawing.XDrawView
- com.sun.star.beans.XPropertySet
- com.sun.star.frame.XController
- com.sun.star.view.XSelectionSupplier

The following methods of com.sun.star.view.XSelectionSupplier control the current selection in the GUI:

```
boolean select( [in] any anObject)
any getSelection()
void addSelectionChangeListener ( [in] com::sun::star::view::XSelectionChangeListener aListen
void removeSelectionChangeListener ( [in] com::sun::star::view::XSelectionChangeListener aListener)
```

With these methods of com.sun.star.drawing.XDrawView, the visible page is set in the GUI:

```
void setCurrentPage(com::sun::star::drawing::XDrawPage aPage)
com::sun::star::drawing::XDrawPage getCurrentPage()
```

In addition to DrawingDocumentDrawView, it supports the following interfaces. For details about these interfaces, refer to *6 Office Development*.

- com.sun.star.task.XStatusIndicatorSupplier
- com.sun.star.ui.XContextMenuInterception
- drafts.com.sun.star.frame.XDispatchInformationProvider
- com.sun.star.frame.XDispatchProvider

9.7.2 Zooming

Zooming is currently not supported through API.

9.7.3 Other Drawing-Specific View Settings

Drawings and presentations can be switched to certain view modes. This is done by the following drawing-specific controller properties of `com.sun.star.drawing.DrawingDocumentDrawView`:

Properties of `com.sun.star.drawing.DrawingDocumentDrawView`	
`IsLayerMode`	`boolean` — Switch to layer mode.
`IsMasterPageMode`	`boolean` — Switch to master page mode.

Furthermore, there are many properties that can be changed through the `XViewDataSupplier` interface of the document:

Methods of `com.sun.star.document.XViewDataSupplier`:

```
com::sun::star::container::XIndexAccess getViewData()
void setViewData( [in] com::sun::star::container::XIndexAccess aData)
```

To use ViewData properties, call `getViewData()` and receive a com.sun.star.container.XIndexContainer:

Methods of `XIndexContainer`:

```
type getElementType()
boolean hasElements()
long getCount()
any getByIndex( [in] long Index)
void replaceByIndex( [in] long Index, any Element)
void insertByIndex( [in] long Index, any Element)
void removeByIndex( [in] long Index)
```

Use `getByIndex()` to iterate over the view data properties, find the required `com.sun.star.beans.PropertyValue` by checking the property names. If found, set the Value Member of the property value and put it back into the container using `replaceByIndex()`. Finally, apply the whole `ViewData` container to the document using `setViewData()`.

The method `setViewData()` is currently not working. It can only be used with `loadComponent-FromURL()`.

10 Charts

10.1 Overview

Chart documents produce graphical representations of numeric data. They are always embedded objects inside other OpenOffice.org documents. The chart document is a document model similar to Writer, Calc and Draw document models, and it can be edited using this document model.

10.2 Handling Chart Documents

10.2.1 Creating Charts

The `com.sun.star.table.XTableChartsSupplier` interface of the `com.sun.star.sheet.Spreadsheet` service is used to create and insert a new chart into a Calc document. This creates a chart that uses data from the `com.sun.star.chart.XChartDataArray` interface of the underlying cell range. A generic way to create charts is to insert an OLE-Shape into a draw page and transform it into a chart setting a class-id.

Creating and Adding a Chart to a Spreadsheet

Charts are used in spreadsheet documents to visualize the data that they contain. A spreadsheet is one single sheet in a spreadsheet document and offers a `com.sun.star.table.XTableChartsSupplier` interface, that is required by the service `com.sun.star.sheet.Spreadsheet`. With this interface, a collection of table charts that are a container for the actual charts can be accessed. To retrieve the chart document model from a table chart object, use the method `getEmbeddedObject()`.

The following example shows how to insert a chart into a spreadsheet document and retrieve its chart document model. The example assumes that there is a `com.sun.star.sheet.XSpreadsheet` to insert the chart and an array of cell range addresses that contain the regions in which the data for the chart can be found. Refer to *8 Spreadsheet Documents* for more information about how to get or create these objects. The following snippet shows how to insert a chart into a Calc document.

```
import com.sun.star.chart.*;
import com.sun.star.uno.UnoRuntime;
import com.sun.star.container.XNameAccess;
import com.sun.star.document.XEmbeddedObjectSupplier;
```

```
final String sChartName = "MyChart";

com.sun.star.table.XTableChartsSupplier aSheet;
com.sun.star.table.CellRangeAddress[] aAddresses;

// get the sheet in which you want to insert a chart
// and query it for XTableChartsSupplier and store it in aSheet
//
// also get an array of CellRangeAddresses containing
// the data you want to visualize and store them in aAddresses
//
// for details see documentation of Spreadsheets

// ...

XChartDocument aChartDocument = null;

com.sun.star.table.XTableCharts aChartCollection = aSheet.getCharts();
XNameAccess aChartCollectionNA = (XNameAccess) UnoRuntime.queryInterface(
    XNameAccess.class, aChartCollection );

// only insert the chart if it does not already exist
if (aChartCollectionNA != null && !aChartCollectionNA.hasByName(sChartName)) {
    // following rectangle parameters are measured in 1/100 mm
    com.sun.star.awt.Rectangle aRect = new com.sun.star.awt.Rectangle(1000, 1000, 15000, 9271);

    // first bool: ColumnHeaders
    // second bool: RowHeaders
    aChartCollection.addNewByName(sChartName, aRect, aAddresses, true, false);

    try {
        com.sun.star.table.XTableChart aTableChart = (com.sun.star.table.XTableChart)
            UnoRuntime.queryInterface(
                com.sun.star.table.XTableChart.class,
                aChartCollectionNA.getByName(sChartName));

        // the table chart is an embedded object which contains the chart document
        aChartDocument = (XChartDocument) UnoRuntime.queryInterface(
            XChartDocument.class,
            ((XEmbeddedObjectSupplier) UnoRuntime.queryInterface(
                XEmbeddedObjectSupplier.class,
                aTableChart )).getEmbeddedObject());

    } catch (com.sun.star.container.NoSuchElementException ex) {
        System.out.println("Couldn't find chart with name " + sChartName + ": " + ex);
    }
}

// now aChartDocument should contain an XChartDocument representing the newly inserted chart
```

Creating a Chart OLE Object in Draw and Impress

The alternative is to create an OLE shape and insert it into a draw page. Writer, Spreadsheet documents and Draw/Impress documents have access to a draw page. The shape is told to be a chart by setting the unique class-id.



The unique Class-Id string for charts is "`12dcae26-281f-416f-a234-c3086127382e`".

A draw page collection is obtained from the `com.sun.star.drawing.XDrawPagesSupplier` of a draw or presentation document. To retrieve a single draw page, use `com.sun.star.container.XIndexAccess`.

A spreadsheet document is also a `com.sun.star.drawing.XDrawPagesSupplier` that provides draw pages for all sheets, that is, the draw page for the third sheet is obtained by calling `getByIndex(2)` on the interface `com.sun.star.container.XIndexAccess` of the container, returned by `com.sun.star.drawing.XDrawPagesSupplier:getDrawPages()`.

A spreadsheet draw page can be acquired directly at the corresponding sheet object. A single sheet supports the service `com.sun.star.sheet.Spreadsheet` that supplies a single page, `com.sun.star.drawing.XDrawPageSupplier`, where the page is acquired using the method `getDrawPage()`.

The OpenOffice.org Writer currently does not support the creation of OLE Charts or Charts based on a text table in a Writer document using the API.

The document model is required once a chart is inserted. In charts inserted as OLE2Shape, the document model is available at the property `Model` of the OLE2Shape after setting the `CLSID`.



Note, that the mechanism described for OLE objects seems to work in Writer, that is, you can see the OLE-Chart inside the Text document after executing the API calls described, but it is not treated as a real OLE object by the Writer. Thus, you can not activate it by double-clicking, because it puts the document into an inconsistent state.

The following example assumes a valid drawing document in the variable `aDrawDoc` and creates a chart in a Draw document. See *9 Drawing* for an example of how to create a drawing document. (Charts/ChartHelper.java)

```
...
final String msChartClassID = "12dcae26-281f-416f-a234-c3086127382e";
com.sun.star.frame.XModel aDrawDoc;

// get a draw document into aDrawDoc
// ...

// this will become the resulting chart
XChartDocument aChartDoc;

com.sun.star.drawing.XDrawPagesSupplier aSupplier = (com.sun.star.drawing.XDrawPagesSupplier)
    UnoRuntime.queryInterface(com.sun.star.drawing.XDrawPagesSupplier.class, aDrawDoc);

com.sun.star.drawing.XShapes aPage = null;
try {
    // get first page
    aPage = (com.sun.star.drawing.XShapes) aSupplier.getDrawPages().getByIndex(0);
} catch (com.sun.star.lang.IndexOutOfBoundsException ex) {
    System.out.println("Document has no pages: " + ex);
}

if (aPage != null) {
    // the document should be a factory that can create shapes
    XMultiServiceFactory aFact = (XMultiServiceFactory) UnoRuntime.queryInterface(
        XMultiServiceFactory.class, aDrawDoc);

    if (aFact != null) {
        try {
            // create an OLE 2 shape
            com.sun.star.drawing.XShape aShape = (com.sun.star.drawing.XShape)
                UnoRuntime.queryInterface(
                    com.sun.star.drawing.XShape.class,
                    aFact.createInstance("com.sun.star.drawing.OLE2Shape"));

            // insert the shape into the page
            aPage.add(aShape);
            aShape.setPosition(new com.sun.star.awt.Point(1000, 1000));
            aShape.setSize(new com.sun.star.awt.Size(15000, 9271));

            // make the OLE shape a chart
            XPropertySet aShapeProp = (XPropertySet) UnoRuntime.queryInterface(
                XPropertySet.class, aShape );

            // set the class id for charts
            aShapeProp.setPropertyValue("CLSID", msChartClassID);

            // retrieve the chart document as model of the OLE shape
            aChartDoc = (XChartDocument) UnoRuntime.queryInterface(
                XChartDocument.class,
                aShapeProp.getPropertyValue("Model"));
        } catch(com.sun.star.uno.Exception ex) {
            System.out.println("Couldn't change the OLE shape into a chart: " + ex);
        }
    }
}
```

Setting the Chart Type

The default when creating a chart is a bar diagram with vertical bars. If a different chart type is required, apply a different diagram type to this initial chart. For example, to create a pie chart, insert the default bar chart and change it to a pie chart.

To change the type of a chart, create an instance of the required diagram service by its service name using the service factory provided by the `com.sun.star.chart.XChartDocument`. This interface is available at the chart document model. After this service instance is created, set it using the `setDiagram()` method of the chart document.

In the following example, we change the chart type to a `com.sun.star.chart.XYDiagram`, also known as a scatter chart. We have assumed that there is a chart document in the variable `aChartDoc` already. The previous sections described how to create a document.

```
// let aChartDoc be a valid XChartDocument

// get the factory that can create diagrams
XMultiServiceFactory aFact = (XMultiServiceFactory) UnoRuntime.queryInterface(
    XMultiServiceFactory.class, aChartDoc);

XDiagram aDiagram = aFact.createInstance("com.sun.star.chart.XYDiagram");

aChartDoc.setDiagram(aDiagram);

// now we have an xy-chart
```

Diagram Service Names
com.sun.star.chart.BarDiagram
com.sun.star.chart.AreaDiagram
com.sun.star.chart.LineDiagram
com.sun.star.chart.PieDiagram
com.sun.star.chart.DonutDiagram
com.sun.star.chart.NetDiagram
com.sun.star.chart.XYDiagram
com.sun.star.chart.StockDiagram

10.2.2 Accessing Existing Chart Documents

To get a container of all charts contained in a spreadsheet document, use the `com.sun.star.table.XTableChartsSupplier` of the service `com.sun.star.sheet.Spreadsheet`, which is available at single spreadsheets.

To get all OLE-shapes of a draw page, use the interface `com.sun.star.drawing.XDrawPage`, that is based on `com.sun.star.container.XIndexAccess`. You can iterate over all shapes on the draw page and check their `CLSID` property to find out, whether the found object is a chart.

10.3 Working with Charts

10.3.1 Document Structure

The important service for charts is `com.sun.star.chart.ChartDocument`. The chart document contains all the top-level graphic objects, such as a legend, up to two titles called `Title` and `Subtitle`, an axis title object for each primary axis if the chart supports axis. The `com.sun.star.chart.ChartArea` always exists. This is the rectangular region covering the complete chart documents background. The important graphical object is the diagram that actually contains the visualized data.

Apart from the graphical objects, the chart document is linked to some data. The required service for the data component is `com.sun.star.chart.ChartData`. It is used for attaching a data change listener and querying general properties of the data source, such as the number to be interpreted as NaN ("not a number"), that is, a missing value. The derived class `com.sun.star.chart.ChartDataArray` allows access to the actual values. Every component providing the `ChartData` service should also support `ChartDataArray`.

The following diagram shows the services contained in a chart and their relationships.

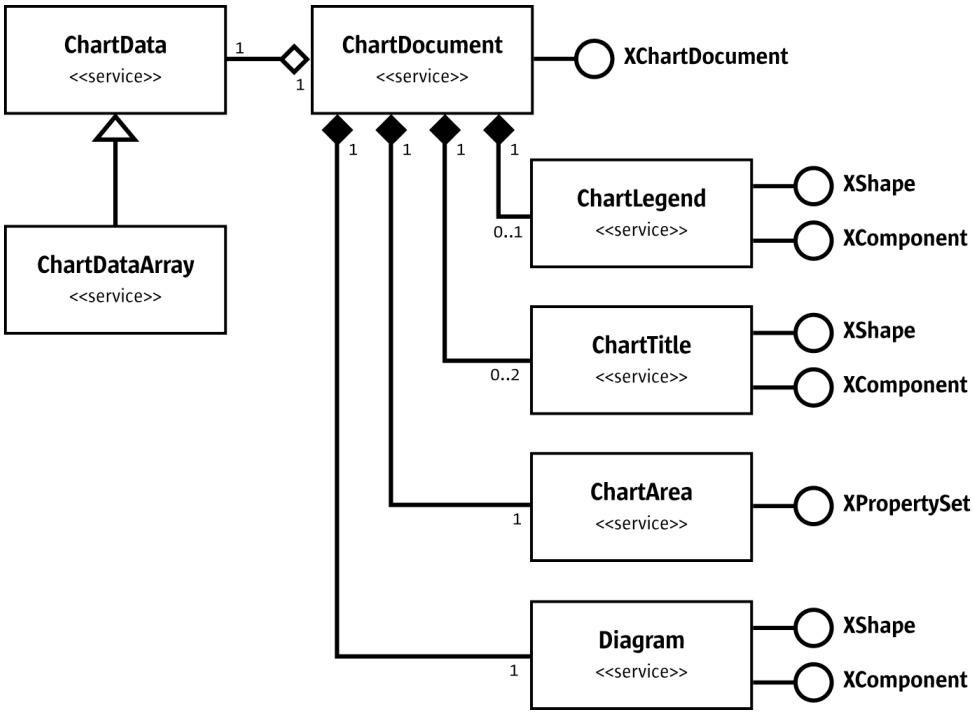


Illustration 139: ChartDocument

The name spaces in the diagram have been omitted to improve readability. The services are all in the name space `com.sun.star.chart`. The interfaces in this diagram are `com.sun.star.chart.XChartDocument`, `com.sun.star.drawing.XShape`, `com.sun.star.lang.XComponent`, and `com.sun.star.beans.XPropertySet`.

The chart document model passes its elements through the interface `com.sun.star.chart.XChartDocument`. This interface consists of the following methods:

```
com::sun::star::chart::XChartData getData()
void attachData( [in] com::sun::star::chart::XChartData xData)
com::sun::star::drawing::XShape getTitle()
```

```
com::sun::star::drawing::XShape getSubTitle()
com::sun::star::drawing::XShape getLegend()
com::sun::star::beans::XPropertySet getArea()
com::sun::star::chart::XDiagram getDiagram()
void setDiagram( [in] com::sun::star::chart::XDiagram xDiagram)

void dispose()
void addEventListener( [in] com::sun::star::lang::XEventListener xListener)
void removeEventListener( [in] com::sun::star::lang::XEventListener aListener)
boolean attachResource( [in] string aURL,
                        [in] sequence <com::sun::star::beans::PropertyValue aArgs)
string getURL()
sequence <com::sun::star::beans::PropertyValue > getArgs()

void connectController( [in] com::sun::star::frame::XController xController)
void disconnectController( [in] com::sun::star::frame::XController xController)
void lockControllers()
void unlockControllers()
boolean hasControllersLocked()
com::sun::star::frame::XController getCurrentController()
void setCurrentController( [in] com::sun::star::frame::XController xController)
com::sun::star::uno::XInterface getCurrentSelection()
```

10.3.2 Data Access

Data can only be accessed for reading when a chart resides inside a spreadsheet document and was inserted as a table chart, that is, the table chart obtains its data from cell ranges of spreadsheets. To change the underlying data, modify the content of the spreadsheet cells. For OLE charts, that is, charts that were inserted as `OLE2Shape` objects, modify the data.

The data of a chart is acquired from the `com.sun.star.chart.XChartDocument` interface of the chart document model using the method `com.sun.star.chart.XChartDocument:getData()`. The current implementation of OpenOffice.org charts provides a `com.sun.star.chart.XChartDataArray` interface, derived from `com.sun.star.chart.XChartData` and supports the service `com.sun.star.chart.ChartDataArray`.



Note that the interface definition of `com.sun.star.chart.XChartDocument` does not require `XChartDocument.getData()` to return an `XChartDataArray`, but `XChartData`, so there may be implementations that do not offer access to the values.

The methods of `XChartDataArray` are:

```
sequence <sequence < double > > getData()
void setData( [in] sequence <sequence < double > > aData)

sequence < string > getRowDescriptions ()
void setRowDescriptions(sequence < string aRowDescriptions >)

sequence < string > getColumnDescriptions()
void setColumnDescriptions(sequence < string aColumnDescriptions >)
```

Included are the following methods from `XChartData`:

```
void addChartDataChangeEventListener(
                [in] com::sun::star::chart::XChartDataChangeEventListener aListener)
void removeChartDataChangeEventListener (
                [in] com::sun::star::chart::XChartDataChangeEventListener aListener)

double getNotANumber()
boolean isNotANumber( [in] double nNumber)
```

The `com.sun.star.chart.XChartDataArray` interface offers several methods to access data. A sequence of sequences is obtained containing `double` values by calling `getData()`. With `setData()`, such an array of values can be applied to the `XChartDataArray`.

The arrays are a nested array, not two-dimensional. Java has only nested arrays, but in Basic a nested array must be used instead of a multidimensional array. The following example shows how to apply values to a chart in Basic:

```
' create data with dimensions 2 x 3
' 2 is called outer dimension and 3 inner dimension

' assume that oChartDocument contains a valid
' com.sun.star.chart.XChartDocument

Dim oData As Object
Dim oDataArray( 0 To 1 ) As Object
Dim oSeries1( 0 To 2 )   As Double
Dim oSeries2( 0 To 2 )   As Double

oSeries1( 0 ) = 3.141
oSeries1( 1 ) = 2.718
oSeries1( 2 ) = 1.0

oSeries2( 0 ) = 17.0
oSeries2( 1 ) = 23.0
oSeries2( 2 ) = 42.0

oDataArray( 0 ) = oSeries1()
oDataArray( 1 ) = oSeries2()

' call getData() method of XChartDocument to get the XChartDataArray
oData = oChartDocument.Data

' call setData() method of XChartDataArray to apply the data
oData.Data = oDataArray()

' Note: to use the series arrays here as values for the series in the chart
'       you have to set the DataRowSource of the XDiagram object to
'       com.sun.star.chart.ChartDataRowSource.ROWS (default is COLUMNS)
```



The `Data` obtianed is a sequence that contains one sequence for each row. If you want to interpret the *inner* sequences as data for the series, set the `DataRowSource` of your `XDiagram` to `com.sun.star.chart.ChartDataRowSource.ROWS`, otherwise, the data for the n^{th} series is in the n^{th} element of each *inner* series.

With the methods of the `XChartData` interface, check if a number from the chart has to be interpreted as non-existent, that is, the number is *not a number* (NaN).



In the current implementation of OpenOffice.org Chart, the value of NaN is not the standard ISO value for NaN of the C++ double type, but instead `DBL_MIN` which is $2.2250738585072014^{-308}$.

Additionally, the `XChartData` interface is used to connect a component as a listener on data changes. For example, to use a spreadsheet as the source of the data of a chart that resides inside a presentation. It can also be used in the opposite direction: the spreadsheet could display the data from a chart that resides in a presentation document. To achieve this, the `com.sun.star.table.CellRange` service also points to the `XChartData` interface, so that a listener can be attached to update the chart OLE object.

The following class `ListenAtCalcRangeInDraw` demonstrates how to synchronize the data of a spreadsheet range with a chart residing in another document. Here the chart is placed into a drawing document.

The class `ListenAtCalcRangeInDraw` in the example below implements a `com.sun.star.lang.XEventListener` to get notified when the spreadsheet document or drawing document are closed.

It also implements a `com.sun.star.chart.XChartDataChangeEventListener` that listens for changes in the underlying `XchartData`. In this case, it is the range in the spreadsheet.

```
import com.sun.star.uno.UnoRuntime;
import com.sun.star.lang.XEventListener;
import com.sun.star.beans.XPropertySet;
import com.sun.star.lang.XComponent;

import com.sun.star.chart.*;
```

```
import com.sun.star.sheet.XSpreadsheetDocument;

// implement an XEventListener for listening to the disposing
// of documents. Also implement XChartDataChangeEventListener
// to get informed about changes of data in a chart

public class ListenAtCalcRangeInDraw implements XChartDataChangeEventListener {
    public ListenAtCalcRangeInDraw(String args[]) {
        // create a spreadsheet document in maSheetDoc
        // create a drawing document in maDrawDoc
        // put a chart into the drawing document
        // and store it in maChartDocument
        // ...

        com.sun.star.table.XCellRange aRange;
        // assign a range from the spreadsheet to aRange
        // ...

        // attach the data coming from the cell range to the chart
        maChartData = (XChartData) UnoRuntime.queryInterface(XChartData.class, aRange);
        maChartDocument.attachData(maChartData);
    }

    public void run() {
        try {
            // show a sub title to inform about last update
            ((XPropertySet) UnoRuntime.queryInterface(
                XPropertySet.class, maChartDocument)).setPropertyValue(
                    "HasSubTitle", new Boolean(true));

            // start listening for death of spreadsheet
            ((XComponent) UnoRuntime.queryInterface(XComponent.class,
                maSheetDoc)).addEventListener(this);

            // start listening for death of chart
            ((XComponent) UnoRuntime.queryInterface(XComponent.class,
                maChartDocument)).addEventListener(this);

            //start listening for change of data
            maChartData.addChartDataChangeEventListener(this);
        } catch (com.sun.star.uno.Exception ex) {
            System.out.println("Oops: " + ex);
        }

        // call listener once for initialization
        ChartDataChangeEvent aEvent = new ChartDataChangeEvent();
        aEvent.Type = ChartDataChangeType.ALL;
        chartDataChanged(aEvent);
    }


    // XEventListener (base interface of XChartDataChangeEventListener)
    public void disposing(com.sun.star.lang.EventObject aSourceObj)
    {
        // test if the Source object is a chart document
        if( UnoRuntime.queryInterface(XChartDocument.class, aSourceObj.Source) != null)
            System.out.println("Disconnecting Listener because Chart was shut down");

        // test if the Source object is a spreadsheet document
        if (UnoRuntime.queryInterface(XSpreadsheetDocument.class, aSourceObj.Source) != null)
            System.out.println("Disconnecting Listener because Spreadsheet was shut down");

        // remove data change listener
        maChartData.removeChartDataChangeEventListener(this);

        // remove dispose listeners
        ((XComponent) UnoRuntime.queryInterface(XComponent.class,
            maSheetDoc)).removeEventListener(this);
        ((XComponent) UnoRuntime.queryInterface(XComponent.class,
            maChartDocument)).removeEventListener(this);
        // this program cannot do anything any more
        System.exit(0);
    }


    // XChartDataChangeEventListener
    public void chartDataChanged(ChartDataChangeEvent aEvent)
    {
        // update subtitle with current date
        String aTitle = new String("Last Update: " + new java.util.Date(System.currentTimeMillis()));

        try {
            ((XPropertySet) UnoRuntime.queryInterface(XPropertySet.class,
                maChartDocument.getSubTitle())).setPropertyValue(
                    "String", aTitle);

            maChartDocument.attachData(maChartData);
```

```
        } catch(Exception ex) {
            System.out.println("Oops: " + ex);
        }

        System.out.println("Data has changed");
    }

    // private
    private com.sun.star.sheet.XSpreadsheetDocument    maSheetDoc;
    private com.sun.star.frame.XModel                  maDrawDoc;
    private com.sun.star.chart.XChartDocument          maChartDocument;
    private com.sun.star.chart.XChartData              maChartData;
}
```

10.3.3 Chart Document Parts

In this section, the parts that most diagram types have in common are discussed. Specific chart types are discussed later.

Common Parts of all Chart Types

Diagram

The diagram object is an important object of a chart document. The diagram represents the visualization of the underlying data. The diagram object is a graphic object group that lies on the same level as the titles and the legend. From the diagram, data rows and data points are obtain that are also graphic objects that represent the respective data. Several properties can be set at a diagram to influence its appearance. Note that the term data series is used instead of data rows.

Some diagrams support the service `com.sun.star.chart.Dim3DDiagram` that contains the property `Dim3D`. If this is set to `true`, you get a three-dimensional view of the chart, which in OpenOffice.org is usually rendered in OpenGL. In 3-D charts, you can access the z-axis, which is not available in 2-D.

The service `com.sun.star.chart.StackableDiagram` offers the properties *Percent* and *Stacked*. With these properties, accumulated values can be stacked for viewing. When setting `Percent` to `true`, all slices through the series are summed up to 100 percent, so that for an `AreaDiagram` the whole diagram space would be filled with the series. Note that setting the `Percent` property also sets the `Stacked` property, because `Percent` is an addition to `Stacked`.

There is a third service that extends a base diagram type for displaying statistical elements called `com.sun.star.chart.ChartStatistics`. With this service, error indicators or a mean value line are added. The mean value line represents the mean of all values of a series. The regression curve is only available for the `XYDiagram`, because a numeric x-axis, is required.

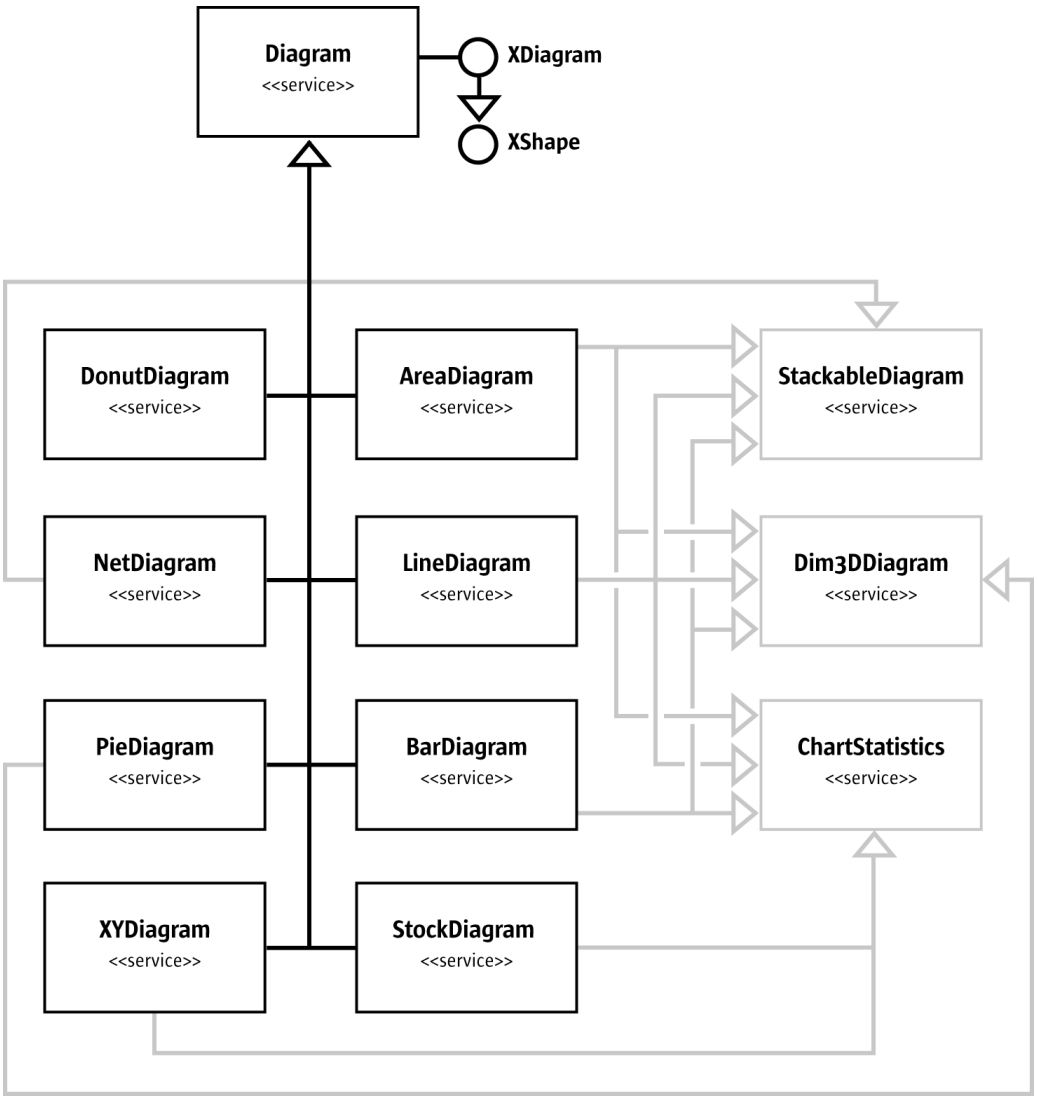


Illustration 140: Diagram

The illustration above shows that there are eight base types of diagrams. The three services, `StackableDiagram`, `Dim3DDiagram` and `ChartStatistics` are also supported for several diagram types and allows extensions of the base types as discussed. For instance, a three-dimensional pie chart can be created, because the `com.sun.star.chart.PieDiagram` service points to the `com.sun.star.chart.Dim3DDiagram` service.

The services `com.sun.star.chart.AreaDiagram`, `com.sun.star.chart.LineDiagram`, and `com.sun.star.chart.BarDiagram` support all three *feature services.*

Axis

All charts can have one or more axis, except for pie charts. A typical two-dimensional chart has two axis, an x- and y-axis. Secondary x- or y-axis can be added to have up to four axis. In a three-dimensional chart, there are typically three axis, x-, y- and z-axis. There are no secondary axis in 3-dimensional charts.

An axis combines two types of properties:

- Scaling properties that affect other objects in the chart. A minimum and maximum values are set that spans the visible area for the displayed data. A step value can also be set that determines the distance between two tick-marks, and the distance between two grid-lines if grids are switched on for the corresponding axis.
- Graphical properties that influence the visual impression. These are character properties (see `com.sun.star.style.CharacterProperties`) affecting the labels and line properties (see `com.sun.star.drawing.LineProperties`) that are applied to the axis line and the tick-marks.

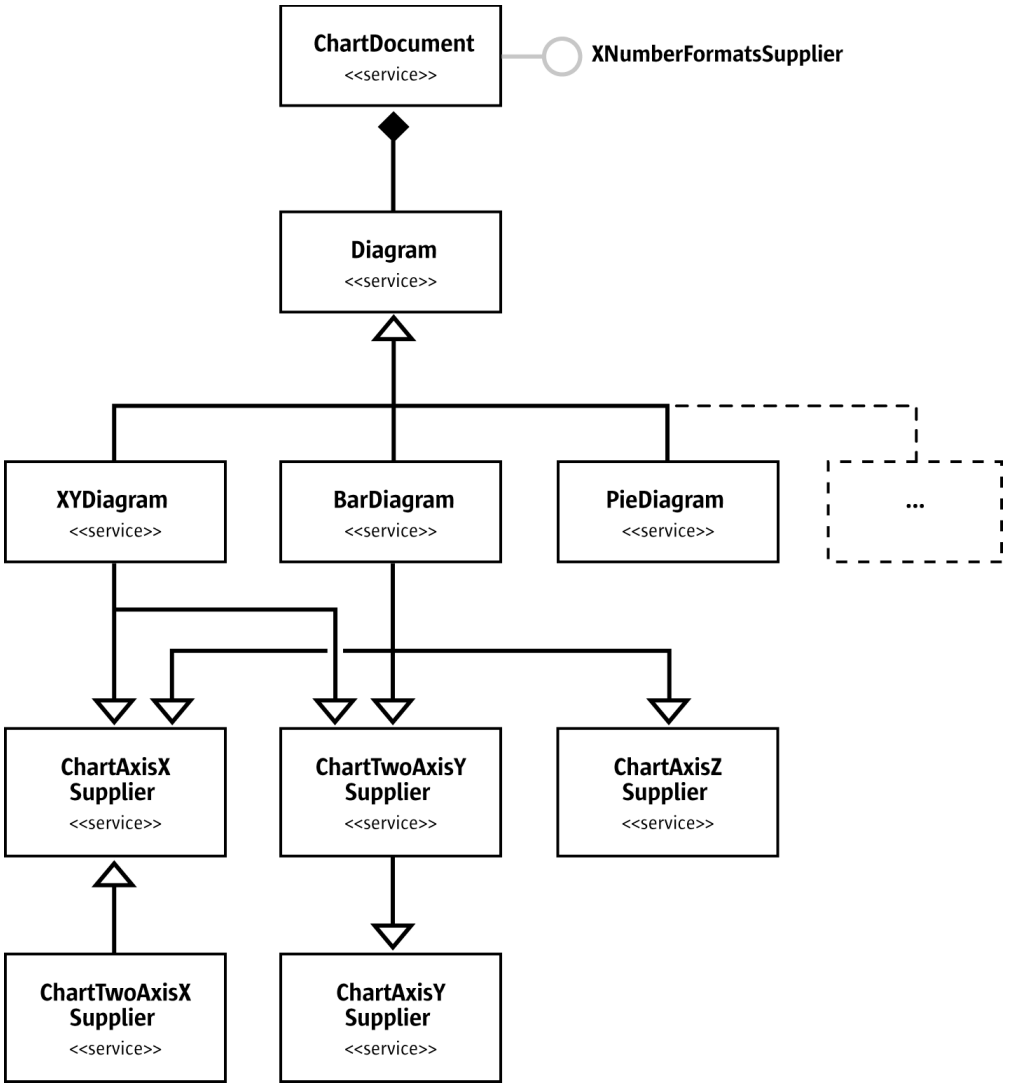


Illustration 141: Axis

Different diagram types support a different number of axis. In the above illustration, a `com.sun.star.chart.XYDiagram`, also known as a scatter diagram, is shown. The scatter diagram supports x- and y-axis, but not a z-axis as there is no 3-dimensional mode. The `com.sun.star.chart.PieDiagram` supports no axis at all. The `com.sun.star.chart.BarDiagram` supports all kinds of axis. Note that the z-Axis is only supported in a three-dimensional chart. Note that there is a `com.sun.star.chart.ChartTwoAxisXSupplier` that includes the `com.sun.star.chart.ChartAxisXSupplier` and is supported by all diagrams in OpenOffice.org required to support the service `ChartAxisXSupplier`.

The following example shows how to obtain an axis and how to change the number format.

```
import com.sun.star.chart.*;
import com.sun.star.beans.XPropertySet;
import com.sun.star.util.XNumberFormatsSupplier;

...

// class members
XChartDocument aChartDocument;
XDiagram aDiagram;

...

// get an XChartDocument and assign it to aChartDocument
// get the diagram from the document and assign it to aDiagram
// ...

// check whether the current chart supports a y-axis
XAxisYSupplier aYAxisSupplier = (XAxisYSupplier) UnoRuntime.queryInterface(
    XAxisYSupplier.class, aDiagram);

if (aYAxisSupplier != null) {
    XPropertySet aAxisProp = aYAxisSupplier.getYAxis();

    // initialize new format with value for standard
    int nNewNumberFormat = 0;

    XNumberFormatsSupplier aNumFmtSupp = (XNumberFormatsSupplier)
        UnoRuntime.queryInterface(XNumberFormatsSupplier.class,
                                  aChartDocument);

    if (aNumFmtSupp != null) {
        com.sun.star.util.XNumberFormats aFormats = aNumFmtSupp.getNumberFormats();

        Locale aLocale = new Locale("de", "DE", "de");

        String aFormatStr = aFormats.generateFormat(
            0,        // base key
            aLocale,  // locale
            true,     // thousands separator on
            true,     // negative values in red
            (short)3, // number of decimal places
            (short)1  // number of leading ciphers
        );

        nNewNumberFormat = aFormats.addNew(aFormatStr, aLocale);
    }

    aAxisProp.setPropertyValue("NumberFormat", new Integer(nNewNumberFormat));
}
```

Data Series and Data Points

The objects that visualize the actual data are data series. The API calls them data rows that are not rows in a two-dimensional spreadsheet table, but as sets of data, because the data for a *data row* can reside in a column of a spreadsheet table.

The data rows contain data points. The following two methods at the `com.sun.star.chart.XDiagram` interface allow changes to the properties of a whole series or single data point:

```
com::sun::star::beans::XPropertySet getDataRowProperties( [in] long nRow)
com::sun::star::beans::XPropertySet getDataPointProperties( [in] long nCol,
                                                            [in] long nRow)
```

The index provided in these methods is 0-based, that is, 0 is the first series. In `XYDiagrams`, the first series has an index 1, because the first array of values contains the x-values of the diagram that is not visualized. This behavior exists for historical reasons.

In a spreadsheet context, the indexes for `getDataPointProperties()` are called `nCol` and `nRow` and are misleading. The `nRow` parameter gives the data row, that is, the series index. The `nCol` gives the index of the data point inside the series, regardless if the series is taken from rows or columns in the underlying table. To get the sixth point of the third series, write `getDataPoint-Properties(5, 2)`.

Data rows and data points have `com.sun.star.drawing.LineProperties` and `com.sun.star.drawing.FillProperties`. They also support

com.sun.star.style.CharacterProperties for text descriptions that can be displayed next to data points.

Properties can be set for symbols and the type of descriptive text desired. With the SymbolType property, one of several predefined symbols can be set. With SymbolBitmapURL, a URL that points to a graphic in a format known by OpenOffice.org can be set that is then used as a symbol in a com.sun.star.chart.LineDiagram or com.sun.star.chart.XYDiagram.

The following example demonstrates how to set properties of a data point. Before implementing this example, create a chart document and diagram of the type XYDiagram.

```
com.sun.star.chart.XChartDocument  aChartDocument;
com.sun.star.chart.XDiagram        aXYDiagram;

// get a chart document and assign it to aChartDocument
// set an "XYDiagram" and remember the diagram in aXYDiagram
// ...

// get the properties of the fifth data point of the first series
// note that index 1 is the first series only in XYDiagrams
try {
    com.sun.star.beans.XPropertySet aPointProp = maDiagram.getDataPointProperties(4, 1);
} catch (com.sun.star.lang.IndexOutOfBoundsException ex) {
    System.out.println("Index is out of bounds: " + ex);
    System.exit(0);
}

try {
    // set a bitmap via URL as symbol for the first series
    aPointProp.setPropertyValue("SymbolType", new Integer(ChartSymbolType.BITMAPURL));
    aPointProp.setPropertyValue("SymbolBitmapURL",
        "http://graphics.openoffice.org/chart/bullet1.gif");

    // add a label text with bold font, bordeaux red 14pt
    aPointProp.setPropertyValue("DataCaption", new Integer(ChartDataCaption.VALUE));
    aPointProp.setPropertyValue("CharHeight", new Float(14.0));
    aPointProp.setPropertyValue("CharColor", new Integer(0x993366));
    aPointProp.setPropertyValue("CharWeight", new Float(FontWeight.BOLD));
} catch (com.sun.star.uno.Exception ex) {
    System.out.println("Exception caught: " + ex);
}
```

Features of Special Chart Types

Examples of some of the services that are not available for all chart types are discussed in this section. Only examples that can be changed in specific chart types only are discussed.

Statistics

Statistical properties like error indicators or regression curves can be applied. The regression curves can only be used for xy-diagrams that have tuples of values for each data point. The following example shows how to add a linear regression curve to an xy-diagram.

Additionally, the mean value line is displayed and error indicators for the standard deviation are applied.

```
    // get the diagram
    // ...

    // get the properties of a single series
    XPropertySet aProp = maDiagram.getDataRowProperties(1)

    // set a linear regression
    aProp.setPropertyValue("RegressionCurves", ChartRegressionCurveType.LINEAR);

    // show a line at y = mean of the series' values
    aProp.setPropertyValue("MeanValue", new Boolean(true));

    // add error indicators in both directions
    // with the length of the standard deviation
    aProp.setPropertyValue("ErrorCategory", ChartErrorCategory.STANDARD_DEVIATION);
    aProp.setPropertyValue("ErrorIndicator", ChartErrorIndicatorType.TOP_AND_BOTTOM);
```

3-D Charts

Some chart types can display a 3-dimensional representation. To switch a chart to 3-dimensional, set the boolean property `Dim3D` of the service `com.sun.star.chart.Dim3DDiagram`.

In addition to this property, bar charts support a property called `Deep` (see service `com.sun.star.chart.BarDiagram`) that arranges the series of a bar chart along the z-axis, which in a chart, points away from the spectator. The service `com.sun.star.chart.Chart3DBarProperties` offers a property `SolidType` to set the style of the data point solids. The solid styles can be selected from cuboids, cylinders, cones, and pyramids with a square base (see constants in `com.sun.star.chart.ChartSolidType`).

The `XDiagram` of a 3-dimensional chart is also a scene object that supports modification of the rotation and light sources. The example below shows how to rotate the scene object and add another light source.

```
// prerequisite: maDiagram contains a valid bar diagram
// ...

import com.sun.star.drawing.HomogenMatrix;
import com.sun.star.drawing.HomogenMatrixLine;
import com.sun.star.chart.X3DDisplay;
import com.sun.star.beans.XPropertySet;

XPropertySet aDiaProp = (XPropertySet) UnoRuntime.queryInterface(XPropertySet.class, maDiagram);
Boolean aTrue = new Boolean(true);

aDiaProp.setPropertyValue("Dim3D", aTrue);
aDiaProp.setPropertyValue("Deep", aTrue);

// from service Chart3DBarProperties:
aDiaProp.setPropertyValue("SolidType", new Integer(
    com.sun.star.chart.ChartSolidType.CYLINDER));

// change floor color to Magenta6
XPropertySet aFloor = ((X3DDisplay) UnoRuntime.queryInterface(
    X3DDisplay.class, maDiagram)).getFloor();
aFloor.setPropertyValue("FillColor", new Integer(0x6b2394));


// rotate the scene using a homogen 4x4 matrix
// ----------------------------------------
HomogenMatrix aMatrix = new HomogenMatrix();
// initialize matrix with identity
HomogenMatrixLine aLines[] = new HomogenMatrixLine[] {
        new HomogenMatrixLine(1.0, 0.0, 0.0, 0.0),
        new HomogenMatrixLine(0.0, 1.0, 0.0, 0.0),
        new HomogenMatrixLine(0.0, 0.0, 1.0, 0.0),
        new HomogenMatrixLine(0.0, 0.0, 0.0, 1.0)
    };

aMatrix.Line1 = aLines[0];
aMatrix.Line2 = aLines[1];
aMatrix.Line3 = aLines[2];
aMatrix.Line4 = aLines[3];

// rotate 10 degrees along the x axis
double fAngle = 10.0;
double fCosX = java.lang.Math.cos(java.lang.Math.PI / 180.0 * fAngle);
double fSinX = java.lang.Math.sin(java.lang.Math.PI / 180.0 * fAngle);

// rotate -20 degrees along the y axis
fAngle = -20.0;
double fCosY = java.lang.Math.cos(java.lang.Math.PI / 180.0 * fAngle);
double fSinY = java.lang.Math.sin(java.lang.Math.PI / 180.0 * fAngle);

// rotate -5 degrees along the z axis
fAngle = -5.0;
double fCosZ = java.lang.Math.cos(java.lang.Math.PI / 180.0 * fAngle);
double fSinZ = java.lang.Math.sin(java.lang.Math.PI / 180.0 * fAngle);

// set the matrix such that it represents all three rotations in the order
// rotate around x axis then around y axis and finally around the z axis
aMatrix.Line1.Column1 =  fCosY *  fCosZ;
aMatrix.Line1.Column2 =  fCosY * -fSinZ;
aMatrix.Line1.Column3 =  fSinY;

aMatrix.Line2.Column1 =  fSinX *  fSinY *  fCosZ +  fCosX *  fSinZ;
aMatrix.Line2.Column2 = -fSinX *  fSinY *  fSinZ +  fCosX *  fCosZ;
aMatrix.Line2.Column3 = -fSinX *  fCosY;
```

```
aMatrix.Line3.Column1 = -fCosX *  fSinY *  fCosZ +  fSinX *  fSinZ;
aMatrix.Line3.Column2 =  fCosX *  fSinY *  fSinZ +  fSinX *  fCosZ;
aMatrix.Line3.Column3 =  fCosX *  fCosY;

aDiaProp.setPropertyValue("D3DTransformMatrix", aMatrix);

// add a red light source
// ----------------------

// in a chart by default only the second (non-specular) light source is switched on
// light source 1 is the only specular light source that is used here

// set direction
com.sun.star.drawing.Direction3D aDirection = new com.sun.star.drawing.Direction3D();

aDirection.DirectionX = -0.75;
aDirection.DirectionY =  0.5;
aDirection.DirectionZ =  0.5;

aDiaProp.setPropertyValue("D3DSceneLightDirection1", aDirection);
aDiaProp.setPropertyValue("D3DSceneLightColor1", new Integer(0xff3333));
aDiaProp.setPropertyValue("D3DSceneLightOn1", new Boolean(true));
```

Refer to *9 Drawing* for additional details about three-dimensional properties.

Pie Charts

Pie charts support the offset of pie segments with the service `com.sun.star.chart.ChartPieSegmentProperties` that has a property `SegmentOffset` to drag pie slices radially from the center up to an amount equal to the radius of the pie. This property reflects a percentage, that is, values can go from 0 to 100.

```
    // ...

    // drag the fourth segment 50% out
    aPointProp = maDiagram.getDataPointProperties(3, 0)
    aPointProp.setPropertyValue("SegmentOffset", 50)
```

Note that the `SegmentOffset` property is not available for donut charts and three-dimensional pie charts.

Stock Charts

A special data structure must be provided when creating stock charts. When a`com.sun.star.chart.StockDiagram` is set as the current chart type, the data is interpreted in a specific manner depending on the properties `Volume` and `UpDown`. The following table shows what semantics are used for the data series.

Volume	UpDown	Series 1	Series 2	Series 3	Series 4	Series 5
false	false	Low	High	Close	-	-
true	false	Volume	Low	High	Close	-
false	true	Open	Low	High	Close	-
true	true	Volume	Open	Low	High	Close

For example, if the property `Volume` is set to `false` and `UpDown` to `true`, the first series is interpreted as the value of the stock when the stock exchange opened, and the fourth series represents the value when the stock exchange closed. The lowest and highest value during the day is represented in series two and three, respectively.

10.4 Chart Document Controller

Although chart document models have a method `getCurrentController()`, this method currently returns null, therefore the chart controller can not be used.

10.5 Chart Add-Ins

Chart types can also be created by developing components that serve as chart types. Existing chart types can be extended by adding additional shapes or modifying the existing shapes. Alternatively, a chart can be created from scratch. If drawing from scratch, it is an empty canvas and all shapes would have to be drawn from scratch.

Chart Add-Ins are actually UNO components, thus, you should be familiar with the chapter *4 Writing UNO Components*.

10.5.1 Prerequisites

The following interfaces must be supported for a component to serve as a chart add-in:

- `com.sun.star.lang.XInitialization`
- `com.sun.star.util.XRefreshable`
- `com.sun.star.lang.XServiceName`
- `com.sun.star.lang.XServiceInfo`
- `com.sun.star.lang.XTypeProvider` to access the add-in interfaces from OpenOffice.org Basic and other interpreted programming languages (optional).

In addition to these interfaces, the following services must be supported and returned in the `getSupportedServiceNames()` method of `com.sun.star.lang.XServiceInfo`):

- `com.sun.star.chart.Diagram`
- A unique service name that identifies the component. This service name has to be returned in the `getServiceName()` method of `com.sun.star.lang.XServiceName`.

10.5.2 How Add-Ins work

The method `initialize()` from the `com.sun.star.lang.XInitialization` interface is the first method that is called for an add-in. It is called directly after it is created by the `com.sun.star.lang.XMultiServiceFactory` provided by the chart document. This method gets the `XChartDocument` object.

When `initialize()` is called, the argument returned is the chart document. Store this as a member to that it can be called later in the `refresh()` call to access all elements of the chart. The following is an example for the `initialize()` method of an add-in written in Java:

```
// XInitialization
public void initialize(Object[] aArguments) throws Exception, RuntimeException {
    if (aArguments.length > 0) {
        // maChartDocument is a member
        // which is set to the parent chart document
        // that is given as first argument
        maChartDocument = (XChartDocument) UnoRuntime.queryInterface(
            XChartDocument.class, aArguments[0]);
```

```
        XPropertySet aDocProp = (XPropertySet) UnoRuntime.queryInterface(
            XPropertySet.class, maChartDocument);
        if (aDocProp != null) {
            // set base diagram which will be extended in refresh()
            aDocProp.setPropertyValue("BaseDiagram", "com.sun.star.chart.XYDiagram");
        }

        // remember the draw page, as it is frequently used by refresh()
        // (this is not necessary but convenient)
        XDrawPageSupplier aPageSupp = (XDrawPageSupplier) UnoRuntime.queryInterface(
            XDrawPageSupplier.class, maChartDocument);
        if( aPageSupp != null )
            maDrawPage = (XDrawPage) UnoRuntime.queryInterface(
                XDrawPage.class, aPageSupp.getDrawPage());
    }
}
```

An important method of an add-in component is `refresh()` from the `com.sun.star.util.XRefreshable`. This method is called every time the chart is rebuilt. A change of data results in a refresh, but also a resizing or changing of a property that affects the layout calls the `refresh()` method. For example, the property `HasLegend` that switches the legend on and off.

To add shapes to the chart, create them once and modify them later during the refresh calls. In the following example, a line is created in `initialize()` and modified during `refresh()`:

```
// XInitialization
public void initialize(Object[] aArguments) throws Exception, RuntimeException {
    // get document and page -- see above
    // ...


        // get a shape factory
        maShapeFactory = ...;

        // create top line
        maTopLine = (XShape) UnoRuntime.queryInterface(
            XShape.class, maShapeFactory.createInstance("com.sun.star.drawing.LineShape"));
        maDrawPage.add(maTopLine);

        // make line red and thicker
        XPropertySet aShapeProp = (XPropertySet)UnoRuntime.queryInterface(
            XPropertySet.class, maTopLine);
        aShapeProp.setPropertyValue("LineColor", new Integer(0xe01010));
        aShapeProp.setPropertyValue("LineWidth", new Integer(50));


        // create bottom line
        maBottomLine = (XShape) UnoRuntime.queryInterface(
            XShape.class, maShapeFactory.createInstance("com.sun.star.drawing.LineShape"));
        maDrawPage.add(maBottomLine);

        // make line green and thicker
        aShapeProp = (XPropertySet) UnoRuntime.queryInterface(
            XPropertySet.class, maBottomLine);
        aShapeProp.setPropertyValue("LineColor", new Integer(0x10e010));
        aShapeProp.setPropertyValue("LineWidth", new Integer(50));
    }

}

// XRefreshable
public void refresh() throws RuntimeException {
    // position lines
    // --------------

    // get data
    XChartDataArray aDataArray = (XChartDataArray) UnoRuntime.queryInterface(
        XChartDataArray.class, maChartDocument.getData());
    double aData[][] = aDataArray.getData();

    // get axes
    XDiagram aDiagram = maChartDocument.getDiagram();
    XShape aXAxis = (XShape) UnoRuntime.queryInterface(
        XShape.class, ((XAxisXSupplier) UnoRuntime.queryInterface(
            XAxisXSupplier.class, aDiagram)).getXAxis());
    XShape aYAxis = (XShape) UnoRuntime.queryInterface(
        XShape.class, ((XAxisYSupplier) UnoRuntime.queryInterface(
            XAxisYSupplier.class, aDiagram)).getYAxis());

    // calculate points for hull
    final int nLength = aData.length;
```

```
    int i, j;
    double fMax, fMin;

    Point aMaxPtSeq[][] = new Point[1][];
    aMaxPtSeq[0] = new Point[nLength];
    Point aMinPtSeq[][] = new Point[1][];
    aMinPtSeq[0] = new Point[nLength];

    for (i = 0; i < nLength; i++) {
        fMin = fMax = aData[i][1];
        for (j = 1; j < aData[i].length; j++) {
            if (aData[i][j] > fMax)
                fMax = aData[i][j];
            else if (aData[i][j] < fMin)
                fMin = aData[i][j];
        }
        aMaxPtSeq[0][i] = new Point(getAxisPosition(aXAxis, aData[i][0], false),
                                    getAxisPosition(aYAxis, fMax, true));
        aMinPtSeq[0][i] = new Point(getAxisPosition(aXAxis, aData[i][0], false),
                                    getAxisPosition(aYAxis, fMin, true));
    }

    // apply point sequences to lines
    try {
        XPropertySet aShapeProp = (XPropertySet) UnoRuntime.queryInterface(
            XPropertySet.class, maTopLine);
        aShapeProp.setPropertyValue("PolyPolygon", aMaxPtSeq);

        aShapeProp = (XPropertySet) UnoRuntime.queryInterface(
            XPropertySet.class, maBottomLine);
        aShapeProp.setPropertyValue("PolyPolygon", aMinPtSeq);
    } catch (Exception ex) {
    }
}

// determine the position of a value along an axis
// bVertical is true for the y-axis and false for the x-axis
private int getAxisPosition(XShape aAxis, double fValue, boolean bVertical) {
    int nResult = 0;

    if (aAxis != null) {
        XPropertySet aAxisProp = (XPropertySet) UnoRuntime.queryInterface(
            XPropertySet.class, aAxis);

        try {
            double fMin, fMax;
            fMin = ((Double) aAxisProp.getPropertyValue("Min")).doubleValue();
            fMax = ((Double) aAxisProp.getPropertyValue("Max")).doubleValue();
            double fRange = fMax - fMin;

            if (fMin <= fValue && fValue <= fMax && fRange != 0) {
                if (bVertical) {
                    // y==0 is at the top, thus take 1.0 - ...
                    nResult = aAxis.getPosition().Y +
                        (int)((double)(aAxis.getSize().Height) * (1.0 - ((fValue - fMin) / fRange)));
                } else {
                    nResult = aAxis.getPosition().X +
                        (int)((double)(aAxis.getSize().Width) * ((fValue - fMin) / fRange));
                }
            }
        } catch (Exception ex) {
        }
    }
    return nResult;
}
```

The subroutine `getAxisPosition()` is a helper to determine the position of a point inside the diagram coordinates. This add-in calculates the maximum and minimum values for each slice of data points, and creates two polygons based on these points.

For an add-in example written in C++, look at the sample addin of the graphics/sch project on *www.openoffice.org*.

10.5.3 How to Apply an Add-In to a Chart Document

There is no method to set an add-in as a chart type for an existing chart in the graphical user interface. To set the chart type, use an API script, for instance, in OpenOffice.org Basic. The following example sets the add-in with service name “com.sun.star.comp.Chart.JavaSampleChartAddIn” at

the current document. To avoid problems, it is advisable to create a chart that has the same type as the one that the add-in sets as `BaseDiagram` type.

```
Sub SetAddIn
Dim oDoc As Object
Dim oSheet As Object
Dim oTableChart As Object
Dim oChart As Object
Dim oAddIn As Object

    ' assume that the current document is a spreadsheet
    oDoc = ThisComponent
    oSheet = oDoc.Sheets( 0 )

    ' assume also that you already added a chart
    ' named "MyChart" on the first sheet
    oTableChart = oSheet.Charts.getByName( "MyChart" )

    If Not IsNull( oTableChart ) Then
        oChart = oTableChart.EmbeddedObject
        If Not IsNull( oChart ) Then
            oAddIn = oChart.createInstance( "com.sun.star.comp.Chart.JavaSampleChartAddIn" )
            If Not IsNull( oAddIn ) Then
                oChart.setDiagram( oAddIn )
            End If
        End If
    End If
End Sub
```



If you want to create an XML-File on your own and want to activate your add-in for a chart; set the attribute `chart:class` of the `chart:chart` element to “add-in” and the attribute `chart:add-in-name` to the service name that uniquely identifies your component.

11 OpenOffice.org Basic and Dialogs

OpenOffice.org provides functionality to create and manage Basic macros and dialogs. The following sections examine the usage of the OpenOffice.org Basic programming environment.

- Section *11.1 Basic and Dialogs - First Steps with OpenOffice.org Basic* guides you through the necessary steps to write OpenOffice.org Basic UNO Programs.
- Section *11.2 Basic and Dialogs - OpenOffice.org Basic IDE* provides a reference to the functionality of the OpenOffice.org Integrated Development Environment (IDE). It describes:
  - The dialogs to manage Basic and dialog libraries.
  - The functionality of the Basic IDE window: the Basic macro editor and debugger, and the Dialog editor.
  - The assignment of macros to events
- Section *11.3 Basic and Dialogs - Features of OpenOffice.org Basic* describes the Basic programming language integrated in OpenOffice.org, including
  - Provides an overview about the general language features built into OpenOffice.org Basic.
  - Extends the UNO language binding chapter *3.4.3 Professional UNO - UNO Language Bindings - OpenOffice.org Basic* by information on how to access the application specific UNO API.
  - Points out threading and rescheduling characteristics of OpenOffice.org Basic that differ from other languages, such as, from Java, which can be important under certain circumstances.
- Section *11.4 Basic and Dialogs - Advanced Library Organization* describes how the Basic library system stores and manages Basic macros and dialogs in OpenOffice.org, and how the user can access libraries and library elements using the appropriate interfaces.
- Section *11.5 Basic and Dialogs - Programming Dialogs and Dialog Controls* describes the toolkit controls used to create dialogs in the dialog editor. In this section the different types of controls and their specific properties are explained in detail.
- Section *11.6 Basic and Dialogs - Creating Dialogs at Runtime* describes how UNO dialogs can be created at runtime without using the dialog editor. This is useful to show dialogs from UNO components. As this is an advanced way to create dialogs, this section goes deeply into the Toolkit interfaces and extends the section *11.5 Basic and Dialogs - Programming Dialogs and Dialog Controls*.
- Section *11.7 Basic and Dialogs - Library File Structure* discusses the various files used by the Basic IDE.
- Section *11.8 Basic and Dialogs - Library Deployment* discusses the automatic deployment of Basic libraries into a local or a shared OpenOffice.org installation.

11.1 First Steps with OpenOffice.org Basic

Step By Step Tutorial

This section provides a tutorial to enable developers to use the Basic IDE. It describes the necessary steps to write and debug a program in the Basic IDE, and to design a Basic dialog. A comprehensive reference of all tools and options can be found at *11.2 Basic and Dialogs - OpenOffice.org Basic IDE*.

Creating a Module in a Standard Library

1. Create a new Writer document and save the document, for example, *FirstStepsBasic.sxw*.
2. Click **Tools – Macro**.

The **Macro** dialog appears. The **Macro from** list shows macro containers where Basic source code (macros) can come from. There is always an **soffice** container for Basic libraries. Additionally each loaded document can contain Basic libraries.

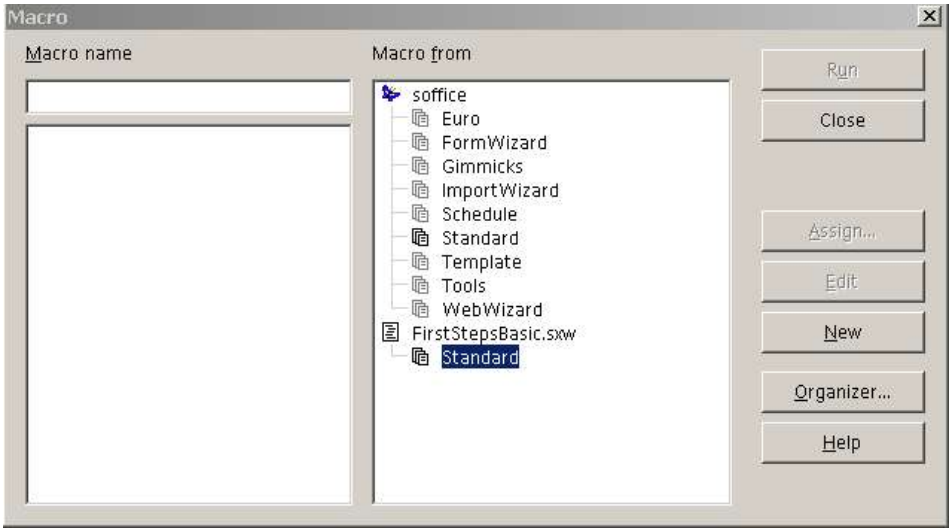


Illustration 142: Macro dialog

The illustration above shows that the document FirstStepsBasic.sxw is the only document loaded. Therefore, the **soffice** and **FirstStepsBasic.sxw** containers are displayed in the illustration above. Both containers, **soffice** and **FirstStepsBasic.sxw,** contain a library named **Standard**. There are a number of other libraries in the **soffice** container that come with a default OpenOffice.org installation – most of them are AutoPilots. The **Standard** libraries of the application and for all open documents are always loaded. They appear enabled in the dialog. Other libraries have to be loaded before they can be used.

The libraries contain modules with the actual Basic source code. Our next step will create a new module for source code in the **Standard** library of our *FirstStepsBasic.sxw* document.

1. Scroll to the document node **FirstStepsBasic.sxw** in the **Macro from** list.
2. Select the **Standard** entry below the document node and click **New**.

OpenOffice.org shows a small dialog that suggests to create a new module named Module1.

1. Click OK to confirm.

The Basic source editor window (Illustration 142) appears containing a Sub (subroutine) Main.

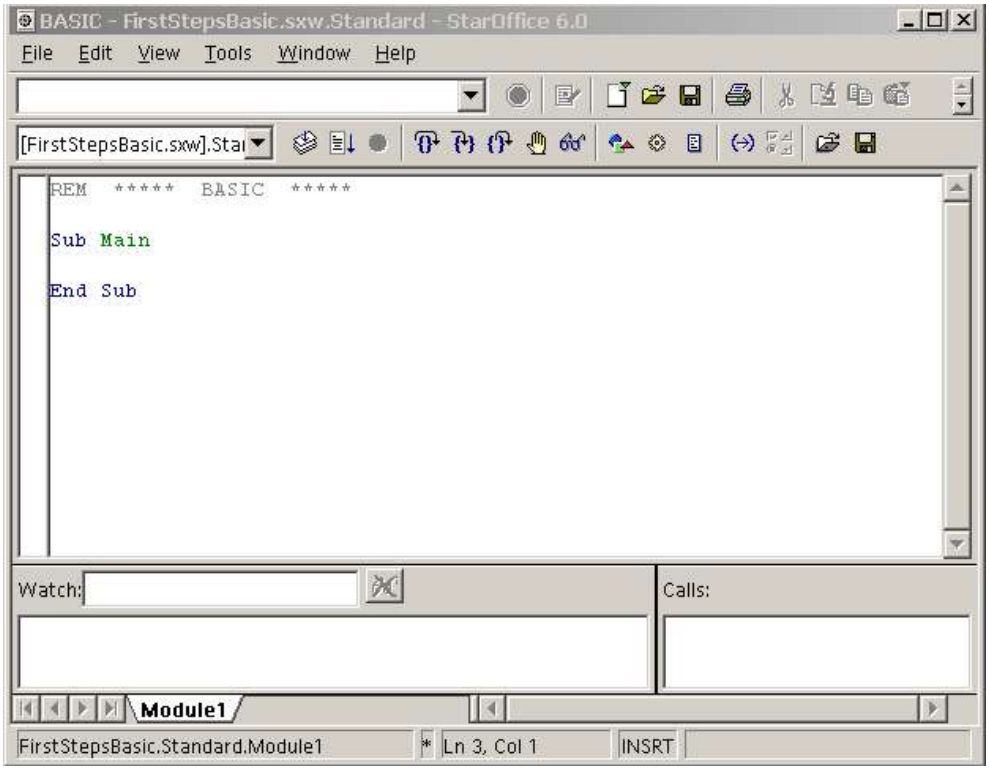


Illustration 143: Basic source editor window

The status bar of the Basic editor window shows that the Sub Main is part of FirstStepsBasic.Standard.Module1. If you click **Tools – Macro** in the Basic editor, you will see that OpenOffice.org created a module **Module1** below the Standard library in *FirstStepsBasic.sxw*.

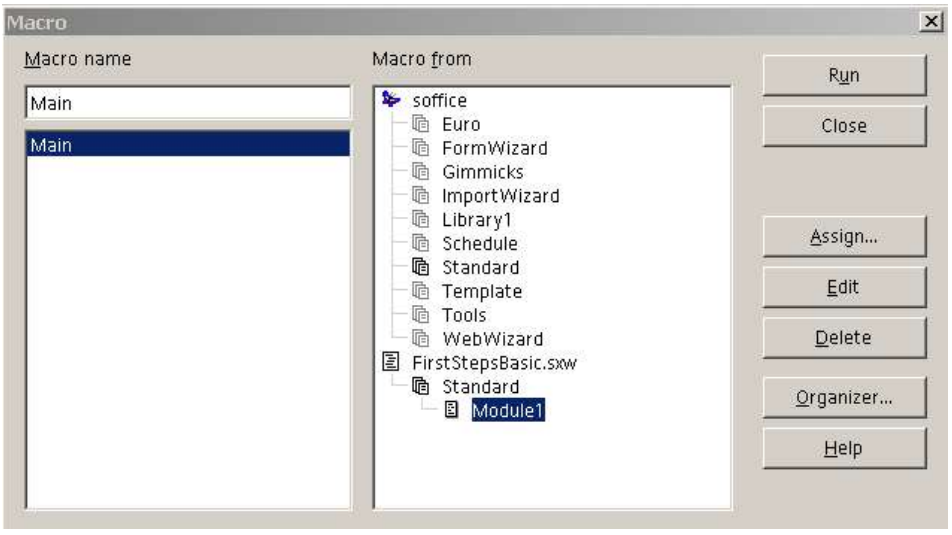


Illustration 144

When a module is selected, the **Macro name** list box on the left shows the Subs and Functions in that module. In this case, Sub Main. If you click **Edit** while a Sub or Function is selected, the Basic editor opens and scrolls to the selected Sub or Function.

Writing and Debugging a Basic UNO program

Enter the following source code in the Basic editor window. The example asks the user for the location of a graphics file and inserts it at the current cursor position of our document. Later, the example will be extended by a small insert graphics autopilot.
(BasicAndDialogs/FirstStepsBasic.sxw)

```
Sub Main
    ' ask the user for a graphics file
    sGraphicUrl = InputBox("Please enter the URL of a graphic file", _
        "Import Graphics", _
        "file:///"
    if sGraphicURL = "" then  ' User clicked Cancel
        exit sub
    endif

    ' access the document model
    oDoc = ThisComponent

    ' get the Text service of the document
    oText = oDoc.getText()

    ' create an instance of a graphic object using the document service factory
    oGraphicObject = oDoc.createInstance("com.sun.star.text.GraphicObject")

    ' set the URL of the graphic
    oGraphicObject.GraphicURL = sGraphicURL

    ' get the current cursor position in the GUI and create a text cursor from it
    oViewCursor = oDoc.getCurrentController().getViewCursor()
    oCursor = oText.createTextCursorByRange(oViewCursor.getStart())

    ' insert the graphical object at the cursor position
    oText.insertTextContent(oCursor.getStart(), oGraphicObject, false)
End Sub
```

If help is required on Basic keywords, press F1 while the text cursor is on a keyword. The OpenOffice.org online help contains descriptions of the Basic language as supported by OpenOffice.org.

Starting with the line `oDoc = ThisComponent`, where the document model is accessed, we use the UNO integration of OpenOffice.org Basic. `ThisComponent` is a shortcut to access a document model from the Basic code contained in it. Earlier, you created Module1 in *FirstStepsBasic.sxw*, that is, your Basic code is embedded in the document *FirstStepsBasic.sxw*, not in a global library below the **soffice** container. The property `ThisComponent` therefore contains the document model of *FirstStepsBasic.sxw*.



Outside document libraries use `ThisComponent` or `StarDesktop.CurrentComponent` to retrieve the current document. If access to an open document is required, even if it is not the current document, you have to iterate over the components in `StarDesktop.Components`, checking their URL property with code similar to the following:

```
oComps = StarDesktop.Components
oCompsEnum = oComps.createEnumeration()

while oCompsEnum.hasMoreElements()
    oComp = oCompsEnum.nextElement()
    ' not all desktop components are necessarily models with a URL
    if HasUnoInterfaces(oComp, "com.sun.star.frame.XModel") then
        print oComp.getURL()
    endif
wend
```



To debug the program, put the cursor into the line `oDoc = ThisComponent` and click the **Breakpoint** icon in the macro bar.



The **Run** icon launches the first Sub in the current module. Execution stops with the first breakpoint.



Now step through the program by clicking the **Single Step** icon.

Click the **Macros** icon if you need to run a Sub other than the first Sub in the module.. In the **Macros** dialog, navigate to the appropriate module, select the Sub to run and press the **Run** button.

To observe the values of simple type Basic variables during debugging, enter a variable name in the **Watch** field of the Basic editor and press the **Enter** key to add the watch, or point at a variable name with the mouse cursor without clicking it. In the example below, we can observe the variable `sGraphicUrl`:

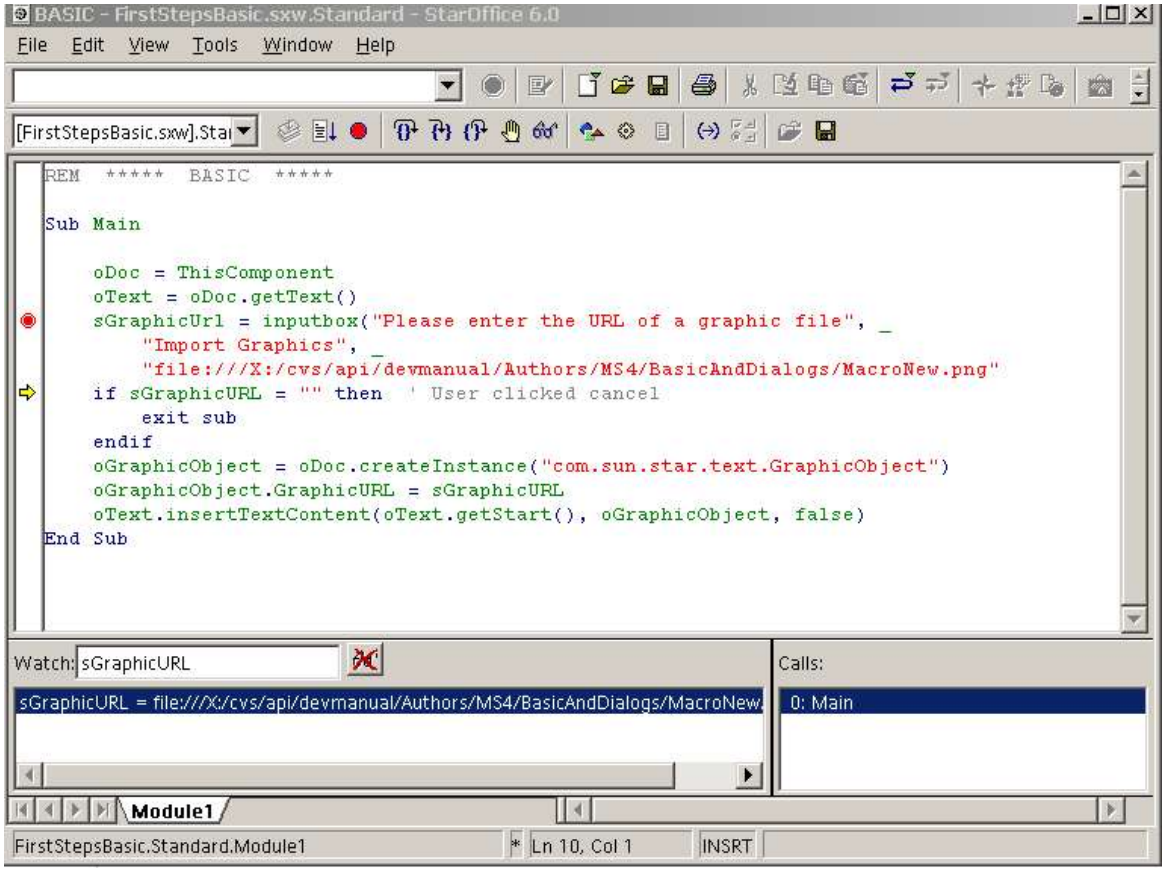


Illustration 145

Currently you can not inspect the values of UNO objects in the Basic debugger during runtime.

Calling a Sub from the User Interface

A Sub can be called from customized icons, menu entries, upon keyboard shortcuts and on certain application or document events. The entry point for all these settings is the **Configuration** dialog accessible through the **Assign...** button in the Macro dialog or the **Tools – Configure** command.

To assign the Sub Main to a toolbar icon, select **Tools – Configure**, click the **Toolbars** tab and click the **Customize** button. The **Customize Toolbars** dialog is displayed.

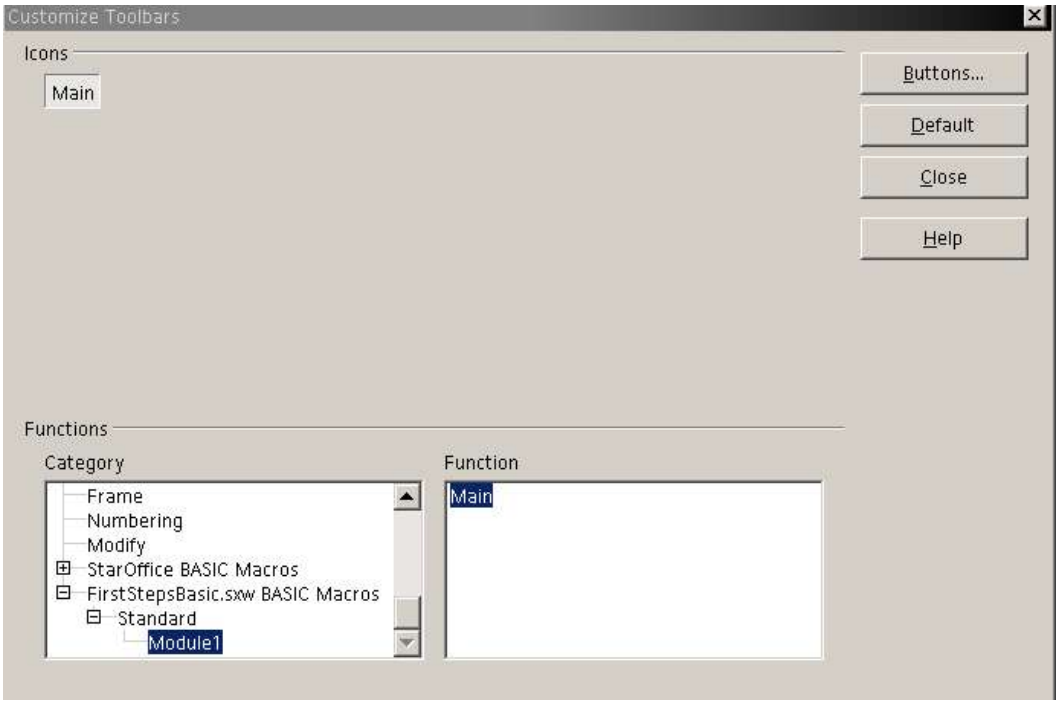


Illustration 146

Scroll down the **Category** list in the **Customize Toolbars** Dialog until you see the Basic libraries. Expand the **FirstStepsBasic.sxw** node. Navigate to the Module FirstStepsBasic.Standard.Module1 and select it. When Module1 is selected, the Icons section shows a button with the caption "Main" for the Sub Main in Module1. The "Main" button can be dragged to a toolbar of your choice. If you want, assign an icon before by clicking **Buttons...** Finally hit **Close** to finish the dialog.

You can now click the new toolbar item to launch the example macros.

The section *11.2.3 Basic and Dialogs - OpenOffice.org Basic IDE - Assigning Macros To GUI Events* describes other options to make your Sub accessible from the user interface.

A Simple Dialog

Creating Dialogs

To create a dialog in the Basic IDE, right-click the Module1 tab at the bottom of the Basic source editor and select **Insert – Basic Dialog**. The IDE creates a new page named Dialog1:

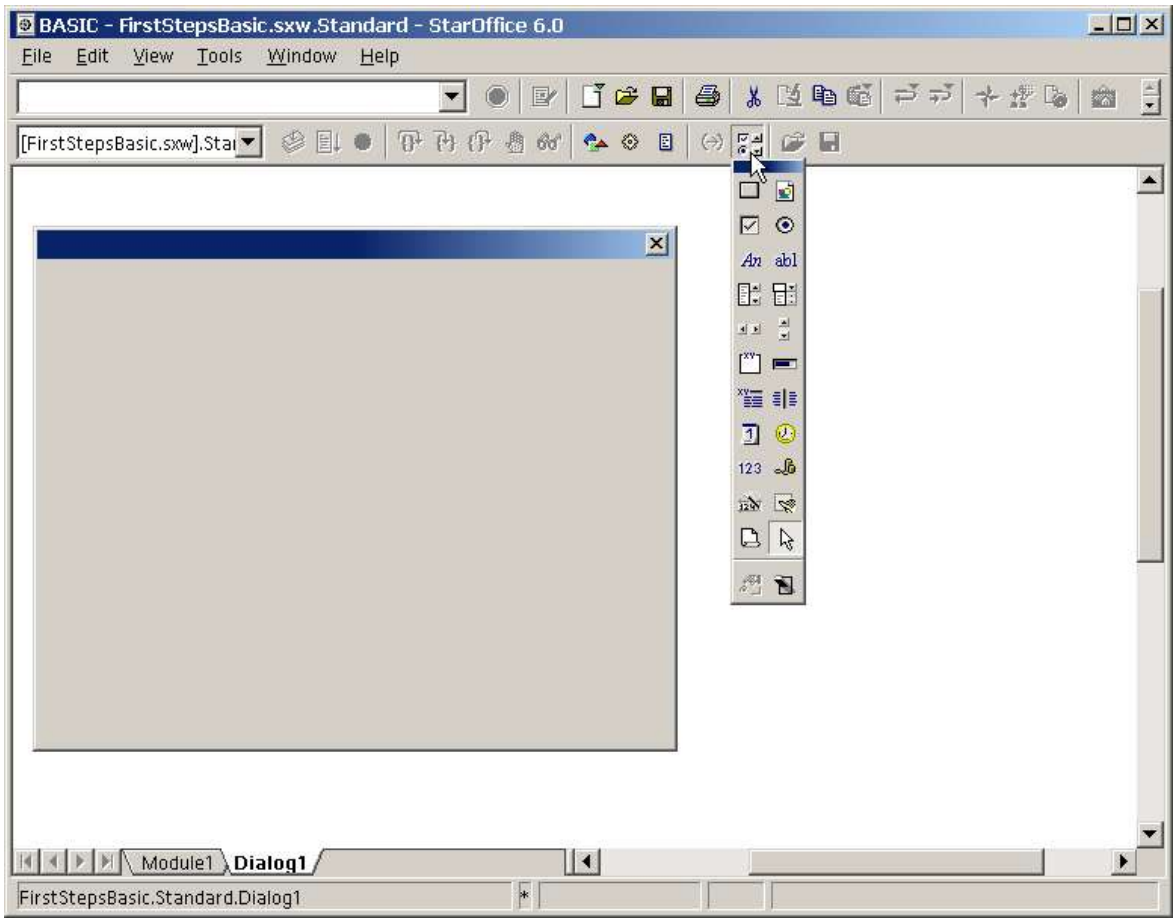


Illustration 147

To add controls to the dialog, we require the dialog design tools. Click the **Controls** icon to pop up the design tools window. The title bar of the tools window can be used to drag the window away from the toolbar to keep it open permanently.

Our dialog shall offer a more convenient way to select a file than the simple input box of our first example. Furthermore, the user shall be able to control how the picture is anchored in the text after inserting it. For this, we will create a wizard dialog with two steps.

In the design tools window, select **File Selection** and define the size of the **Browse** control by dragging a rectangle in the dialog using the left-mouse button.

The **Properties** icon displays the **Properties Dialog** that is used to edit controls and hook up event handling code to events occurring at dialog controls.

Next, add **<< Back** and **Next >>** Buttons to move between the dialog steps, and a **Finish** and **Cancel** button. Select the **Button** icon and define the button size using the left-mouse button. Buttons are labeled with a default text, such as CommandButton1. If the **Properties Dialog** is not open, double click the newly inserted button controls to display it. Enter new labels in the **Label** field as suggested, and name the dialog step buttons `Back` and `Next`. Set the property **Enabled** for the **<< Back** button to false.

Use the **Label** tool to create a label "Select Graphics File:" in the same manner.

Now the dialog looks similar to the illustration below:

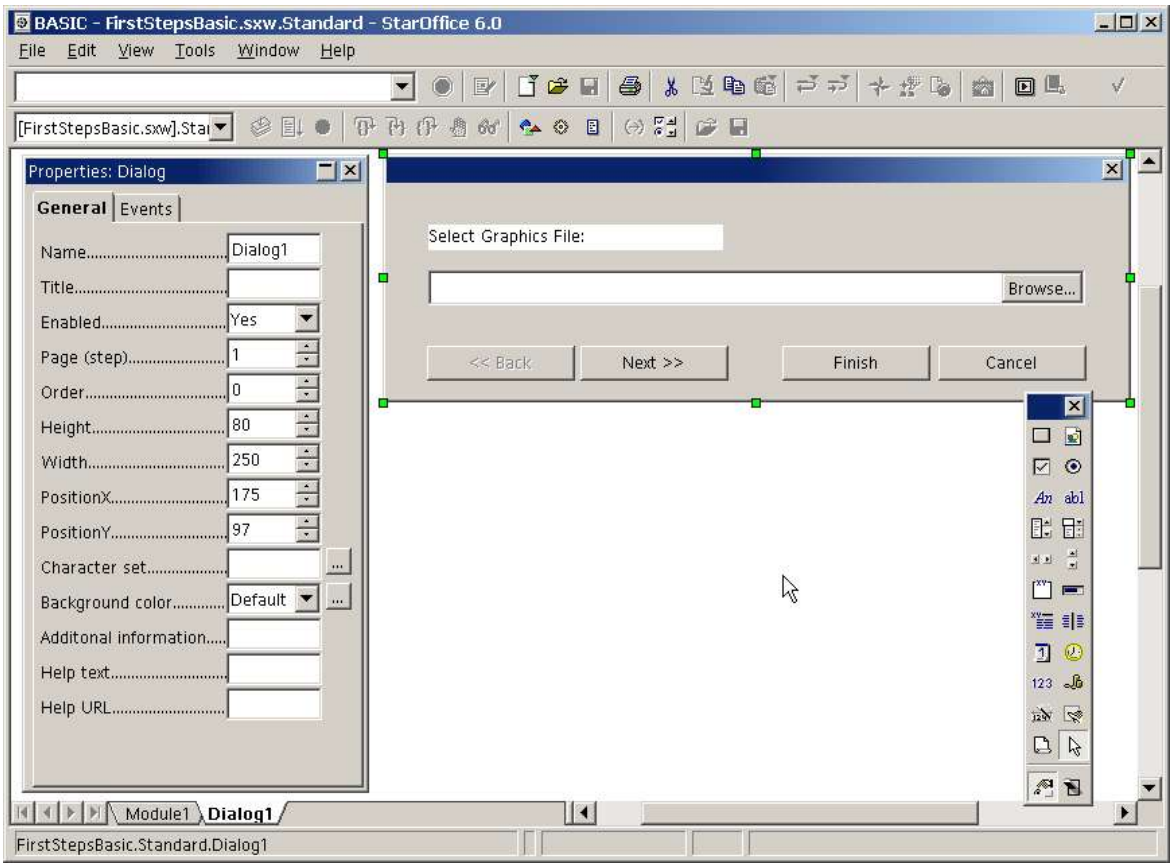


Illustration 148

Test the dialog using the **Activate Test Mode** icon from the design tool window. After you have finished the test, click the **Close** button of the test dialog window.

To edit the dialog, such as setting the title and changing the size, select it by clicking the outer border of the dialog. Green handles appear around the dialog. The green handles can be used to alter the dialog size.The **Properties Dialog** is used to define a dialog title and other dialog properties.

Adding Event Handlers

Now we will write code to open the dialog and add functionality to the buttons. To show a dialog, create a dialog object using `createUnoDialog()` and call its `execute()` method. A dialog can be closed while it is shown by calling `endExecute()`.



It is possible to configure the **Finish** button and the **Cancel** button to close the dialog by setting the button property `PushButtonType` to OK and Cancel respectively. The method `execute()` returns 0 for Cancel and 1 for OK.

To add functionality to GUI elements, develop Subs to handle GUI events, then hook them to the GUI elements. To add functionality to the buttons of our dialog, click the **Module1** tab in the lower part of the Basic IDE and enter the following Subs above the previous Sub Main to open, close and process the dialog. Note that a `Private` variable `oDialog` is defined outside of the Subs. After loading the dialog, this variable is visible from all Subs and Functions of Module1. (BasicAndDialogs/FirstStepsBasic.sxw)

```
Private oDialog as Variant  ' private, module-wide variable
```

```
Sub RunGraphicsWizard
    oDialog = createUnoDialog(DialogLibraries.Standard.Dialog1)
    oDialog.execute
End Sub

Sub CancelGraphicsDialog
    oDialog.endExecute()
End Sub

Sub FinishGraphicsDialog
    Dim sFile as String, sGraphicURL as String

    oDialog.endExecute()

    sFile = oDialog.Model.FileControl1.Text

    ' the FileControl contains a system path, we have to transform it to a file URL
    ' We use the built-in Basic runtime function ConvertToURL for this purpose
    sGraphicURL = ConvertToURL(sFile)

    ' insert the graphics
    ' access the document model
    oDoc = ThisComponent
    ' get the Text service of the document
    oText = oDoc.getText()
    ' create an instance of a graphic object using the document service factory
    oGraphicObject = oDoc.createInstance("com.sun.star.text.GraphicObject")
    ' set the URL of the graphic
    oGraphicObject.GraphicURL = sGraphicURL
    ' get the current cursor position in the GUI and create a text cursor from it
    oViewCursor = oDoc.getCurrentController().getViewCursor()
    oCursor = oText.createTextCursorByRange(oViewCursor.getStart())
    ' insert the graphical object at the cursor position
    oText.insertTextContent(oCursor.getStart(), oGraphicObject, false)
End Sub

Sub Main
        ...
End Sub
```

Select the **Cancel** button in our dialog in the dialog editor, and click the **Events** tab of the **Properties Dialog,** then click the ellipsis**...** button on the right-hand side of the Event **When Initiating**. In the **Assign Macro** dialog, navigate to **FirstStepsBasic.sxw.Standard.Module1**, select the Sub `CancelGraphicsDialog` and click the **Assign** button to link this Sub to the **Cancel** button.

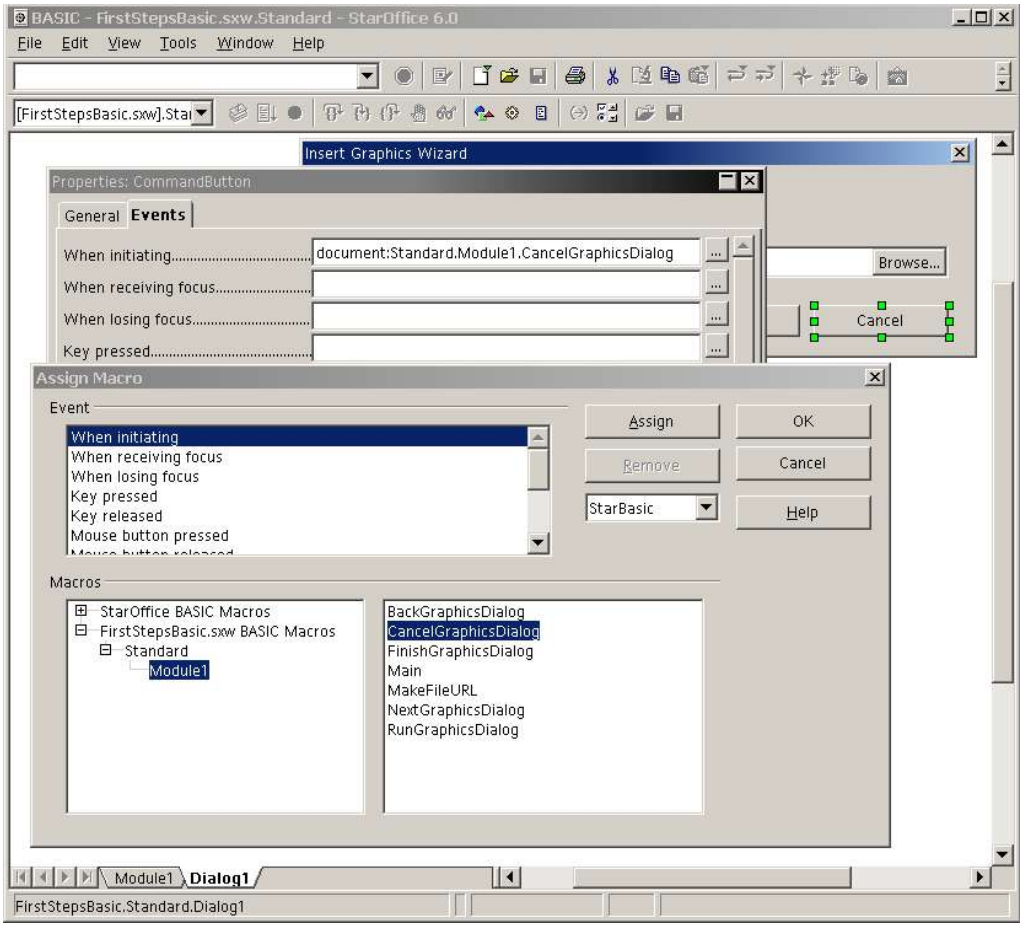


Illustration 149

Using the same method, hook the **Finish** button to `FinishGraphicsDialog`.

If the **Run** icon is selected now, the dialog is displayed, and the **Finish** and **Cancel** buttons are functional.

AutoPilot Dialogs

The final step is to create a small AutoPilot with two pages. The OpenOffice.org Dialogs have a simple concept for AutoPilot pages. Each dialog and each control in a dialog has a property **Page (Step)** to control the pages of a dialog. Normally, dialogs are on page 0, but they can be set to a different page, for example, to page 1. All controls having 1 in their **Page** property are visible as long as the dialog is on page 1. All controls having 2 in their page property are only displayed on page 2 and so forth. If the dialog is on Page 0, all controls are visible at once. If a control has its Page property set to 0, it is visible on all dialog pages.

This feature is used to create a second page in our dialog. Hold down the **Control** key, and click the label and file control in the dialog to select them. In the **Properties Dialog**, fill in 1 for the **Page** property and press **Enter** to apply the change. Next, select the dialog by clicking the outer rim of the dialog in the dialog editor, enter 2 for the **Page** property and press the **Enter** key. The label and file control disappear, because we are on page 2 now. Only the buttons are visible since they are on page 0.

On page 2, add a label "Anchor" and two option buttons "at Paragraph" and "as Character". Name the option buttons `AtParagraph` and `AsCharacter`, and toggle the **State** property of the `AtPara-`

graph button, so that it is selected by default. The new controls automatically receive 2 in their **Page** property. When page 2 is finished, set the dialog to page 1 again, because we want it to be on page 1 on startup.

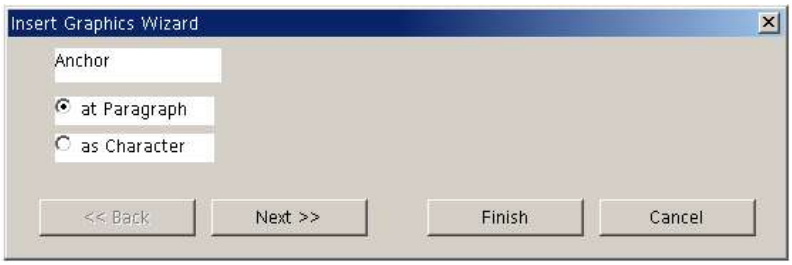


Illustration 150

The Subs below handle the **<< Back** and **Next >>** buttons, and the Sub `FinishGraphicsDialog` has been extended to anchor the new graphics selected by the user. Note that the property that is called **Page (Step)** in the GUI, is called `Step` in the API. (BasicAndDialogs/FirstStepsBasic.sxw)

```
Sub BackGraphicsDialog
    oDialog.Model.Step = 1
    oDialog.Model.Back.Enabled = false
    oDialog.Model.Next.Enabled = true
End Sub

Sub NextGraphicsDialog
    oDialog.Model.Step = 2
    oDialog.Model.Back.Enabled = true
    oDialog.Model.Next.Enabled = false
End Sub

Sub FinishGraphicsDialog
    Dim sGraphicURL as String, iAnchor as Long
    oDialog.endExecute()
    sFile = oDialog.Model.FileControl1.Text

    ' State = Selected corresponds to 1 in the API
    if oDialog.Model.AsCharacter.State = 1 then
        iAnchor = com.sun.star.text.TextContentAnchorType.AS_CHARACTER
    elseif oDialog.Model.AtParagraph.State = 1 then
        iAnchor = com.sun.star.text.TextContentAnchorType.AT_PARAGRAPH
    endif
    ' the File Selection control returns a system path, we have to transform it to a File URL
    ' We use a small helper function MakeFileURL for this purpose (see below)
    sGraphicURL = MakeFileURL(sFile)
    ' access the document model
    oDoc = ThisComponent
    ' get the Text service of the document
    oText = oDoc.getText()
    ' create an instance of a graphic object using the document service factory
    oGraphicObject = oDoc.createInstance("com.sun.star.text.GraphicObject")
    ' set the URL of the graphic
    oGraphicObject.GraphicURL = sGraphicURL
    oGraphicObject.AnchorType = iAnchor
    ' get the current cursor position in the GUI and create a text cursor from it
    oViewCursor = oDoc.getCurrentController().getViewCursor()
    oCursor = oText.createTextCursorByRange(oViewCursor.getStart())
    ' insert the graphical object at the beginning of the text
    oText.insertTextContent(oCursor.getStart(), oGraphicObject, false)
End Sub
```

11.2 OpenOffice.org Basic IDE

This section discusses all features of the *Integrated Development Environment* (IDE) for OpenOffice.org Basic. It shows how to manage Basic and dialog libraries, discusses the tools of the Basic IDE used to create Basic macros and dialogs, and it treats the various possibilities to assign Basic macros to events.

11.2.1 Managing Basic and Dialog Libraries

The main entry point to the library management UI is the **Tools - Macro...** menu item. This item activates the Macro dialog where the user can manage all operations related to Basic and dialog libraries.

Macro Dialog

The following picture shows an example macro dialog. From here you can run, create, edit and delete macros, assign macros to UI events, and administer Basic libraries and modules.

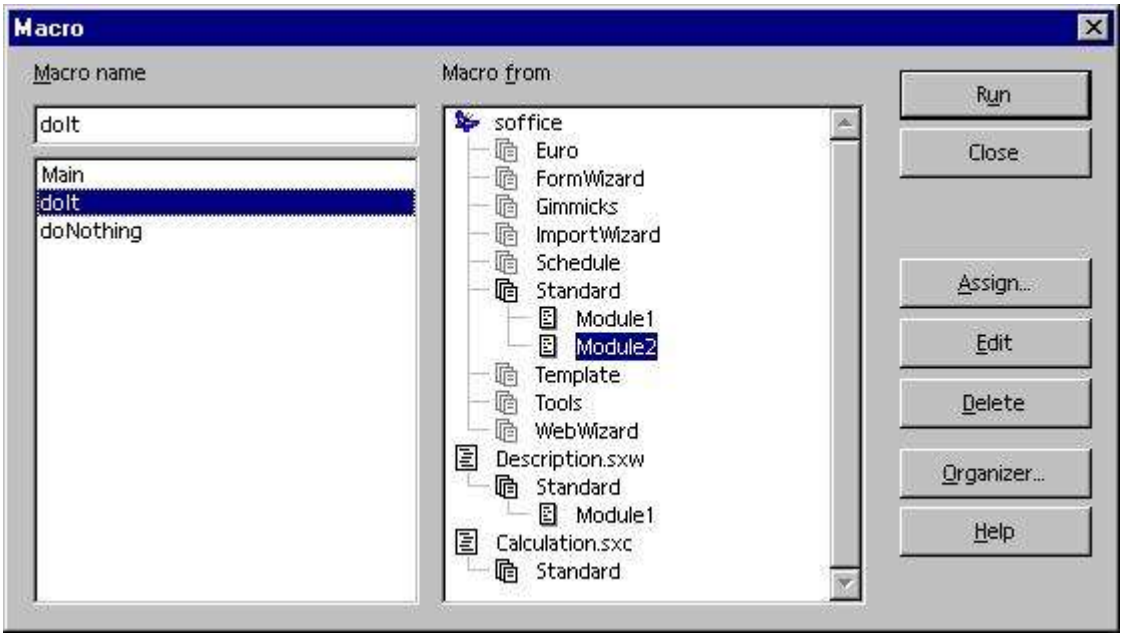


Illustration 151

Displayed Information

The tree titled with **Macro from** shows the complete library hierarchy that is available the moment the dialog is opened. See *11.4 Basic and Dialogs - Advanced Library Organization* for details about the library organization in OpenOffice.org..

Unlike the library organization API, this dialog does not distinguish between Basic and dialog libraries. Usually the libraries displayed in the tree are both Basic and dialog libraries.



Although it is possible to create Basic-only or dialog-only libraries using the APIthis is not the normal case, because the graphical user interface (see *11.2.1 Basic and Dialogs - OpenOffice.org Basic IDE - Managing Basic and Dialog Libraries - Macro Organizer Dialog* below) only allows the creation of Basic and dialog libraries simultaneously. Nevertheless, the dialog can also deal with Basic-only or dialog-only libraries, but they are not marked in any way.

The tree titled **Macro from** represents a structure consisting of three levels:

Library container -> library -> library element

- The top-level nodes represent the application Basic and dialog library container (node `soffice`). Foreach opened document, the document's Basic and dialog library container (see

11.4 Basic and Dialogs - Advanced Library Organization). In the example two documents are open, a text document called *Description.sxw* and a spreadsheet document named *Calculation.sxc*.

- In the second level, each node represents a library. Initially all libraries, except the default libraries named `Standard`, are not loaded and grayed out. To load a library, the user double-clicks the library. In the example above, the `soffice` root element contains the `Standard` library, already loaded by default, and the libraries Euro, Form-Wizard, Gimmicks, ImportWizard, Schedule, Template, Tools, and WebWizard. These libraries represent or belong to the wizards available in the **File - Autopilot** menu.
- The third level in the tree is visible in loaded libraries. Each node represents a library element that can be modules or dialogs. In the macro dialog, only Basic modules are displayed as library elements, whereas dialogs are not shown. By double-clicking a library the user can expand and condense a library to show or hide its modules. In the example, the `soffice/Standard` library is displayed expanded. It contains two modules, `Module1` and `Module2`. The document *Description.sxw* contains a `Standard` library with one Basic module `Module1`. *Calculation.sxc* contains a `Standard` library without Basic modules. All libraries, respectively their dialog library part, may also contain dialogs that cannot be seen in this view.

If a library is password-protected and a user double-clicks it to load it, a dialog is displayed requesting a password. The library is only loaded and expanded if the user enters the correct password. If a password-protected library is loaded using the API, for example, through a call to `BasicLibraries.loadLibrary("Library1")`, it is displayed as loaded, not grayed out, but it remains condensed until the correct password is entered (see *11.4 Basic and Dialogs - Advanced Library Organization*).

Initially all root nodes, the `soffice` and document nodes, are condensed and the contained libraries are displayed. Similar to expanding and condensing libraries, a complete root node can be expanded and condensed as well.

The left column contains information about the macros, that is, the Subs and Functions, in the libraries. In the list box at the bottom, all Subs and Functions belonging to the module selected in the tree are listed. In the edit field titled **Macro name,** the Sub or Function currently selected in the list box is displayed. If there is no module selected in the tree, the edit field and list are empty. You can type in a desired name in the edit field.

Buttons

On the right-hand side of the **Macro** dialog, there are several buttons. The following list describes the buttons:

- **Run**
  Executes the Sub or Function currently displayed in the Macro name edit field. The macro dialog is closed, before the macro is executed.
- **Close**
  Closes the Macro dialog without any further action.
- **Assign**
  Opens the Configuration dialog that can also be opened using **Tools - Configure...**. This dialog can be used to assign Basic macros to events. For details see *11.2.3 Basic and Dialogs - OpenOffice.org Basic IDE - Assigning Macros To GUI Events* below.
- **Edit**
  Loads the module selected in the tree into the Basic macro editor. The cursor is placed on the first line of the Sub or Function displayed in the **Macro name** edit field. See chapter *11.2.2 Basic and Dialogs - OpenOffice.org Basic IDE - Basic IDE Window* below ror details about the Basic

macro editor. This button is disabled if there is no module selected in the tree or no existing Sub or Function displayed in the **Macro name** edit field.

- **Delete**
  This button is only available if an existing Sub or Function is displayed in the **Macro name** edit field. The **Delete** button removes the Sub or Function displayed in the **Macro name** edit field from the module selected in the module selected in the tree.
- **New**
  This button is only available if no existing Sub or Function is displayed in the **Macro name** edit field. The **New** button inserts a new Sub into the module selected in the tree. The new Sub is named according to the text in the **Macro name** edit field. If **Macro name** is empty, the Sub is automatically named Macro1, Macro2, and so forth.
- **Organizer...**
  This button opens the **Macro Organizer** dialog box that is explained in the next section.
- **Help**
  Starts the OpenOffice.org help system with the Macros topic.

Macro Organizer Dialog

This dialog is opened by clicking the Button **Organizer...** in the Macro dialog. The dialog contains the two tab pages **Modules** and **Libraries**. While the Macro dialog refers to Subs and Functions inside Basic modules, such as run Subs, delete Subs, and insert new Subs, this dialog accesses the library system on module/dialog (tab page **Modules**) and library (tab page **Libraries**) level.

Modules

Illustration 148 shows the **Macro Organizer** dialog with the **Modules** tab page activated. The list titled **Module/Dialog** is similar to the **Macro from** list in the **Macro** dialog, but it contains the complete library hierarchy for the OpenOffice.org application libraries and the document libraries. The libraries are loaded, and condensed or expaned by double-clicking the library. The only difference is that the dialogs are listed together with the Basic modules in the **Macro Organizer**. The illustration shows the application library `Standard` containing two modules, Module1 and Module2, and three dialogs, Dialog1, Dialog2 and Dialog3.

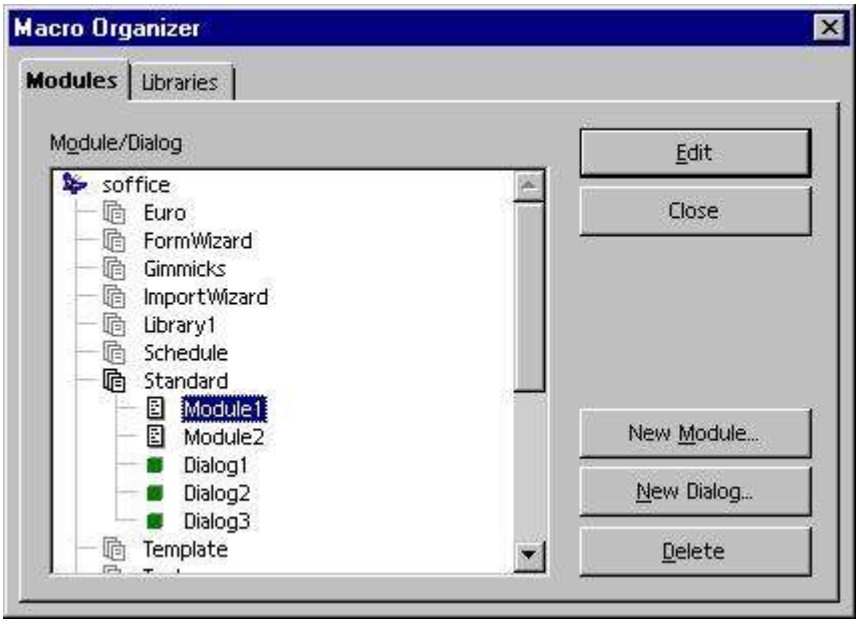


Illustration 152

Illustration 147 presents the same dialog with the **Module/Dialog** listbox that has been scrolled down to show the documents and their libraries.

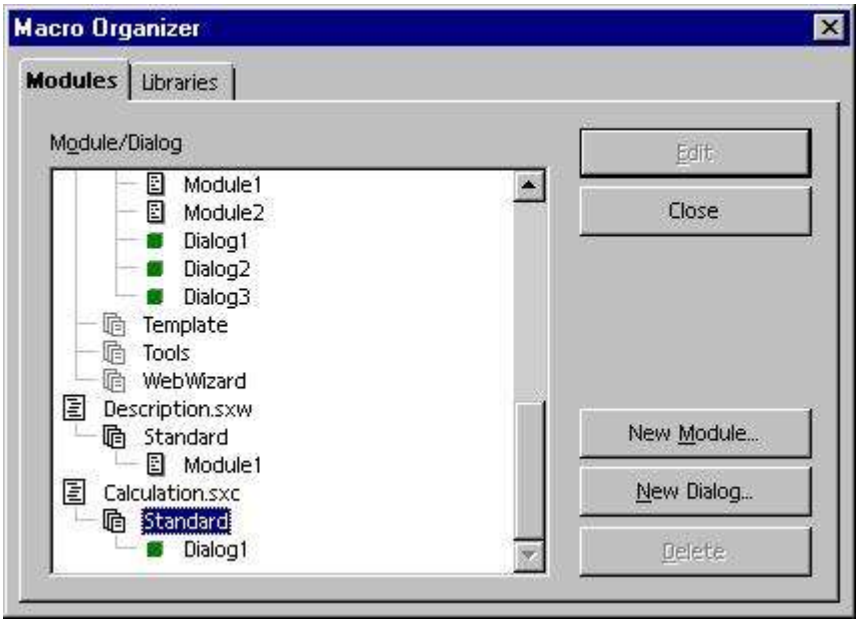


Illustration 153

The illustration above shows that two documents are loaded. The illustration shows a library `Standard` in document *Calculation.sxc* containing a dialog named `Dialog1`, and another library `Standard` in document *Description.sxw* containing a Basic module.

The following list describes the buttons on the right side of the dialog:

- **Edit**
  Loads the module selected in the tree into the Basic macro editor. Also, if a dialog is selected, the Edit button loads the module into the Dialog editor. The section *11.2.2 Basic and Dialogs -*

OpenOffice.org Basic IDE - Basic IDE Window - Dialog Editor belowdescribes the Dialog Editor in more detail. If a module or dialog is not selected, this button is disabled.

- **Close**
  Closes the **Macro organizer** dialog without any further action.
- **New Module...**
  Opens a dialog that allows the user to type in the desired name for a new module. The name edit field initially contains a name like Module<Number>, Such as Module1 and Module2. depending on the modules already existing. Clicking the **OK** button add the new module as a new item in the **Module/Dialog** list. The **New Module...** button is disabled if the selected library has read-only status.
- **New Dialog...**
  Opens a dialog that allows the user to enter the desired name for a new dialog. The name edit field initially contains the name Dialog<Number>, such as Dialog1 and Dialog2, depending on the dialogs already existing. Clicking the **OK** button creates the dialog in the **Module/Dialog** list. This button is disabled if the selection contains a library with read-only status.
- **Delete**
  Deletes the selected module or dialog. This button is disabled if no module or dialog is selected, or if the selected module or dialog belongs to a library with read-only status.

Libraries

The following illustrations show the **Macro Organizer** dialog with the **Libraries** tab page activated. In this dialog, the application and document libraries are listed separately. The **Library** list only contains the libraries of the library container currently selected in the **Application/Document** listbox. The second illustration is dropped down showing the soffice entry and the two open documents.

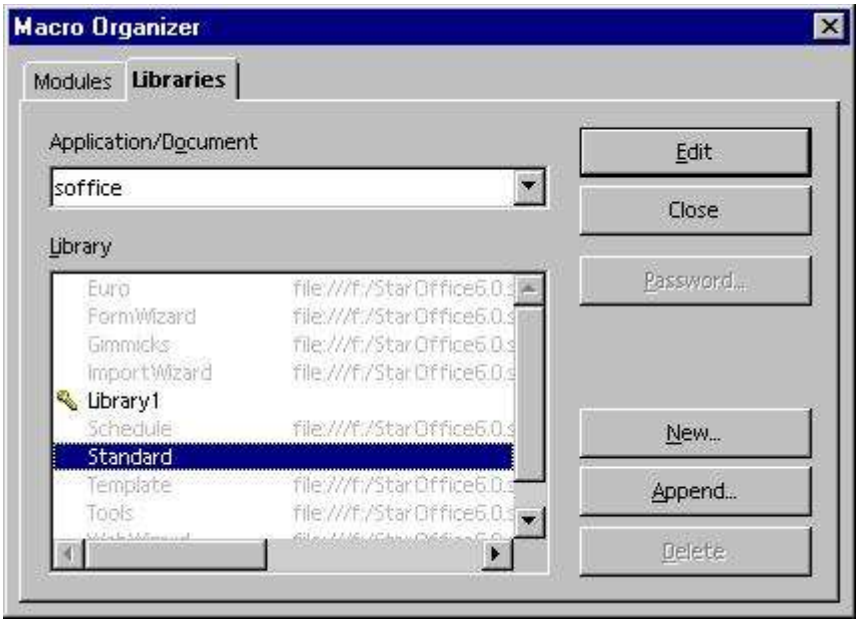


Illustration 154

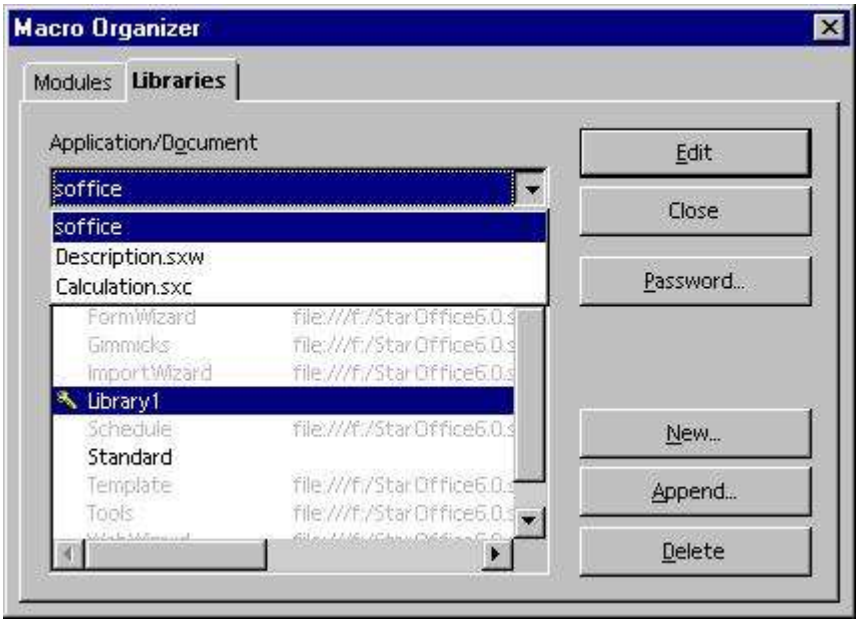


Illustration 155

The libraries are displayed in the following manner:

- Regular libraries are displayed in black.
- Libraries with read-only status are grayed out.
- Library links are followed by an URL indicating the location where the library is stored. In the example above, all libraries except for Standard and Library1 are library links and all library links have read-only status.
- Password protected libraries are indicated with a key symbol before the name. In the example, only Library1 is password protected.

Clicking a library twice (notdouble-click) allows the user to rename it.

The following list describes the buttons on the right side of the dialog:

- **Edit**
  Loads the first module of the library selected in the **Library** listbox into the Basic macro editor (see *11.2.2 Basic and Dialogs - OpenOffice.org Basic IDE - Basic IDE Window - Basic Source Editor and Debugger* below). If the library only contains dialogs, the first dialog of the corresponding dialog library is displayed in the Dialog editor (see *11.2.2 Basic and Dialogs - OpenOffice.org Basic IDE - Basic IDE Window - Dialog Editor* below). If the Basic/Dialog editor window does not exist, it is opened.
- **Close**
  Closes the **Macro Organizer** dialog without any further action.
- **Password...**
  Opens the **Change Password** dialog displayed in the next illustration for the library currently selected in the **Library** listbox.

  This dialog is used to change the password if the library is already password protected. Enter the old password first, then the new password twice.

  If the library is not password protected, the **Old password** edit field is disabled. The new password is entered twice in the **New password** section. Clicking **OK** activates the password protection if both passwords match.

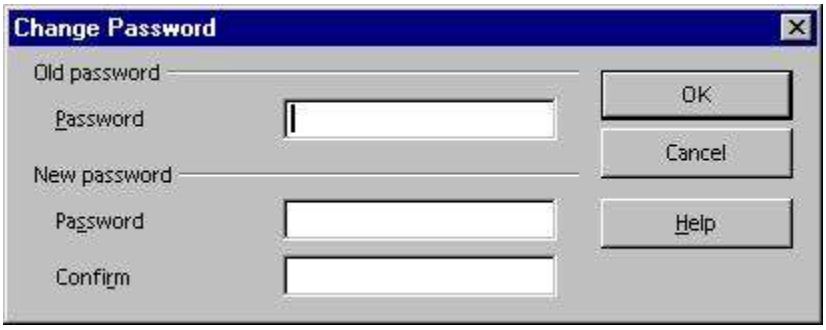


Illustration 156

- **New...**
  Opens a dialog allowing the user to enter the name for a new library. The name edit field initially contains the name Library<Number>, such as Library1 and Library2, depending on the libraries already existing. Clicking the **OK** button creates the library and adds it to the Library list. A new library is always created as a Basic and dialog library.

- **Append...**
  This button is used to import additional libraries into the library container that is selected in the **Application/Document** listbox. The button opens a file dialog where the user selects the location where the libraryis imported from. The following types of files can be selected:

  - Library container index files (*script.xlc* or *dialog.xlc*)
  - Library index files (*script.xlb* or *dialog.xlb*)
  - OpenOffice.org documents (**.sxw*, **.sxc*)
  - Star Office 5.x and previous documents (**.sdw*, **.sdc*)
  - Star Office 5.x and previous Basic library files (**.sbl*)

  After selecting a file, an **Append library** dialog is displayed. The next illustration shows the dialog after selecting a library index file *script.xlb*. The dialog displays all libraries that are found in the chosen file. In this example, only the library Euro appears, because the file *script.xlb* only represented this library.



Illustration 157

The checkboxes in the Options section, when selected, indicatesif a library is inserted as a read-only link and if existing libraries with the same name are replaced by the new library.

The next illustration shows the dialog after selecting the writer document *LibraryImportExample.sxw*. This document contains the four libraries Standard, Library1, Library2 and Library3. The illustration shows that the libraries Library1 and Library2 are selected for import. The **Insert as reference (read-only)** option is disabled, because the libraries inside documents cannot be referenced as a link. As well, StarOffice 5.x Basic libaray files can not be linked.



Illustration 158

Clicking the **OK** button imports the selected libraries into the library container that was previously selected in the **Application/Document** listbox, including the Basic and dialog libraries.

- **Delete**
  Deletes the item selected in the **Library** listbox. If the item represents a library link, only the link is removed, not the library itself. The **Delete** button appears disabled whenever a Standard library is selected, because Standard libraries cannot be deleted.

11.2.2 Basic IDE Window

The OpenOffice.org IDE is mainly represented by the Basic IDE window. The IDE window has two different modes:

- The Basic editor mode displays and modifies Basic source code modules to control the debugging process and display the debugger output
- The dialog editor mode displays and modifies dialogs.

Basic source code and dialogs are never displayed at the same time. The IDE window is in Basic editor or debugger, or in dialog editor mode. The following illustration shows the Basic IDE window in the Basic editor mode displaying Module2 of the application Standard library.

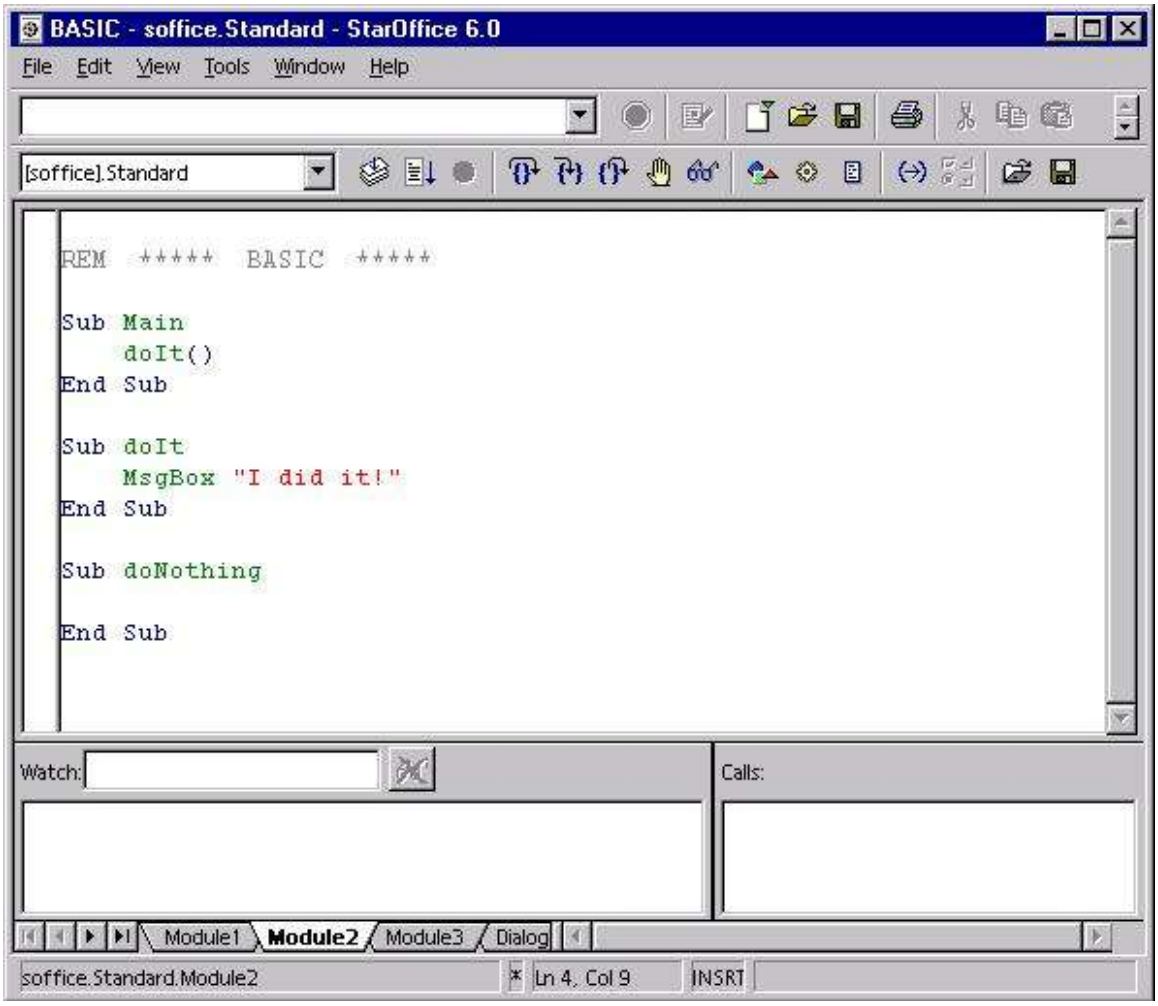


Illustration 159

The IDE window control elements common to the Basic editor and dialog editor mode are described below. The mode specific control elements are described in the corresponding subchapters *11.2.2 Basic and Dialogs - OpenOffice.org Basic IDE - Basic IDE Window - Basic Source Editor and Debugger* and *11.2.2 Basic and Dialogs - OpenOffice.org Basic IDE - Basic IDE Window - Dialog Editor*:

- Clicking the **Printer** button in the main toolbar prints the displayed Basic module or dialog directly without displaying a printer dialog.
- The **Save** button in the main toolbar behaves in two different ways depending on the library currently displayed in the IDE window.
    - If the library belongs to the application library container, the **Save** button saves all modified application libraries.
    - If the library belongs to a document, the **Save** button saves the document.
- On the left-hand side of the toolbar, a **Library** listbox shows the currently displayed library. The user can also modify the displayed library. In the example above, the Standard library of the application Basic ([soffice].Standard) is displayed. The listbox contains all the application and document libraries that are currently accessible. The user can select one to display it in the IDE window.

- The tabs at the bottom of the IDE window indicate all the modules and dialogs of the active library. Clicking one of these tabs activates the corresponding module or dialog. If necessary, the IDE window switches from Basic editor to dialog editor mode or conversely. Right-clicking a tab opens a context menu:
  - **Insert** opens a sub menu to insert a new module or dialog.
  - **Delete** deletes the active module or dialog after confirmation by the user.
  - **Rename** changes the name of the active module or dialog.
  - **Hide** makes the active module or dialog invisible. It no longer appears as a tab flag, thus it cannot be activated. To activate, access it directly using the **Macro** or **Macro Organizer** dialog and clicking the **Edit** button.
  - **Modules...** opens the **Macro Organizer** dialog.
- The status bar displays the following information:
  - The first cell on the left displays the fully qualified name of the active module or dialog in the notation LibraryContainer.LibraryName.<ModuleName | DialogName>.
  - The second cell displays an asterisk "*" indicating that at least one of the libraries of the active library container has been modified and requires saving.
  - The third cell displays the current position of the cursor in the Basic editor window.
  - The fourth cell displays "INSRT" if the Basic editor is in insertion mode and "OVER" if the Basic editor is in overwrite mode. The modes are toggled with the **Insert** key.

Basic Source Editor and Debugger

The Basic editor and debugger of the IDE window is shown when the user edits a Sub or Function from the Tools-Macro dialog (see Illustration 146). In this mode, the window contains the actual editor main window, debugger Watch window to display variable values and the debugger Calls window to display the Basic call stack. The Watch and Calls windows are only used when a Basic program is running and halted by the debugger.

The editor supports common editor features. Since the editor is only used for the OpenOffice.org Basic programming language, it supports a Basic syntax specific highlighting and F1 help for Basic keywords.

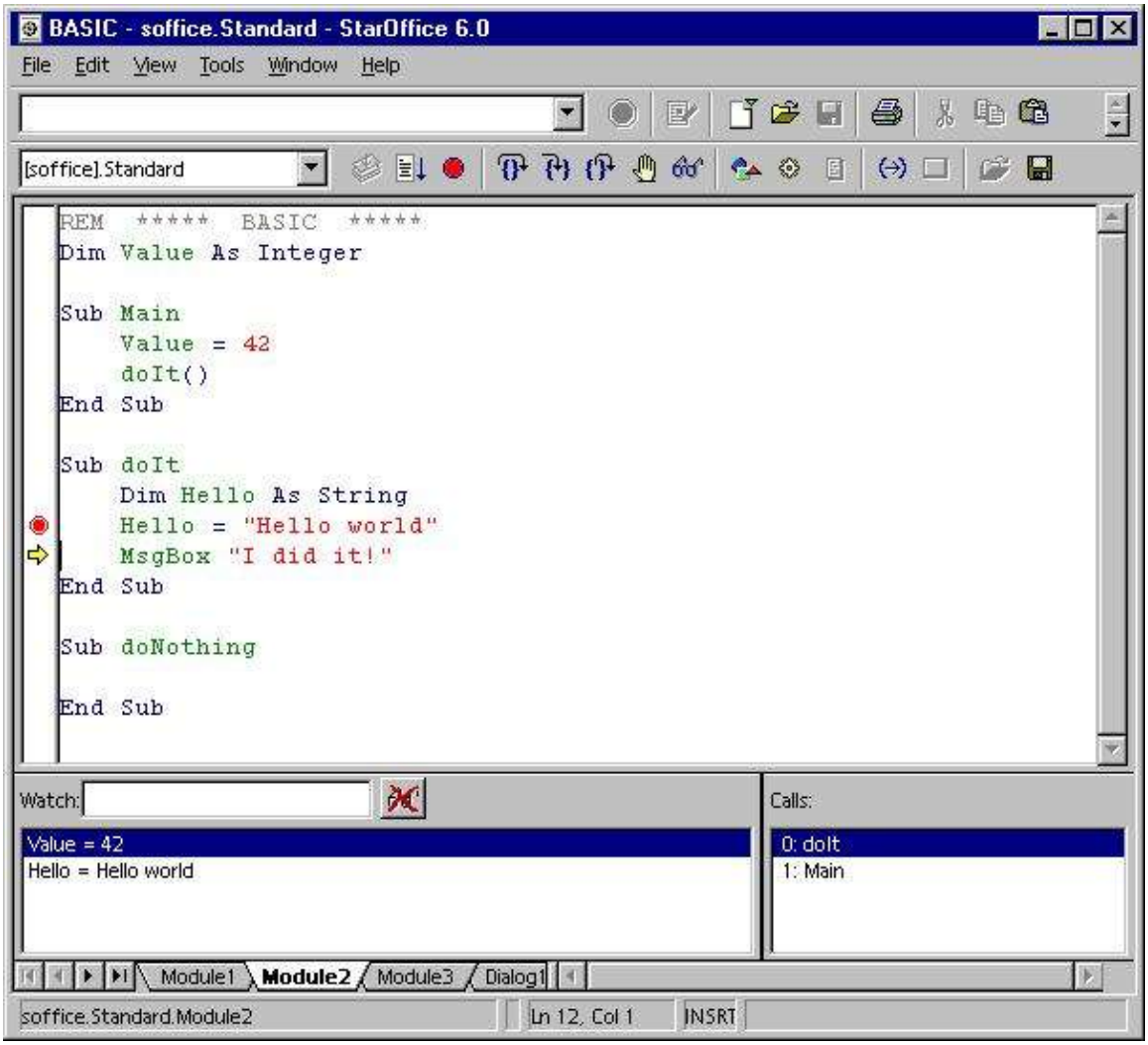


Illustration 160: Basic Editor and Debugger

The following list explains the functionality of the macro toolbar buttons.

Compile: Compiles the active module and displays an error message, if necessary. This button is disabled if a Basic program is running. Always compile libraries before distributing them.

Run: Executes the active module, starting with the first Sub in the module, before all modified modules of the active library are compiled. Clicking this button can also result in compiler errors before the program is started. This button resumes the execution if the program is halted by the debugger.

Stop: Stops the Basic program execution. This button is disabled if a program is not running.

Procedure Step: Executes one Basic statement without stepping into Subs or Functions called in the statement. The execution is halted after the statement has been executed. If the Basic program not is running the execution is started and halted at the first statement of the first Sub in the current module.

Single Step: Executes one Basic statement. If the statement contains another Sub, execution is halted at the first statement of the called Sub. If no Subs or Functions are called in the statement, this button has the same functionality as the **Step over** button (key command **F8**).

Step back: Steps out of the current executed Sub or Function and halts at the next statement of the caller Sub or Function. If the currently executed Sub or Function was not called by another Sub or Function or if the Basic program is not running, this button has the same effect as the **Run** button.

Breakpoint: Toggles a breakpoint at the current cursor line in the Basic editor. If a breakpoint can not be set at this line a beep warns the user and the action is ignored (key command **F9**). A breakpoint is displayed as a red dot in the left column of the editor window.

Add watch: Adds the identifier currently touched by the cursor in the Basic editor to the watch window (key command **F7**).

Object Catalog: Opens the **Objects** dialog. This dialog displays the complete library hierarchy including dialogs, modules and the Subs inside the modules.

Macros: Opens the **Macro Dialog**.

Modules: Opens the **Macro Organizer** dialog

Find Parentheses: If the cursor in the Basic editor is placed before a parenthesis, the matching parenthesis is searched. If a matching parenthesis is found, the code between the two parentheses is selected, otherwise the user is warned by a beep.

Controls: Opens the dialog editing tools in the dialog editor. In Basic editor mode this button is disabled.

Insert Source File: Displays a file open dialog and inserts the selected text file (**.bas* is the standard extension) at the current cursor position into the active module.

Save Source As: Displays a file Save As dialog to save the active module as a text file (**.bas* is the standard extension).

Illustration 146 shows how the IDE window looks while a Basic program is executed in debugging mode.

- The **Stop** button is enabled.
- A breakpoint is set in line 11.
- The execution is halted in line 12. The current position is marked by a yellow arrow.
- The **Watch** window contains the entries Value and Hello, and displays the current values of these variables. Values of variables can also be evaluated by touching a corresponding identifier in the source code with the cursor.
- The **Calls** window shows the stack. The currently executed Sub `doIt` is displayed at the top and the Sub `Main` at the second position.

Dialog Editor

This section provides an overview of the Dialog editor functionality. The controls that are used to design a dialog are not explained. See *11.5 Basic and Dialogs - Programming Dialogs and Dialog Controls* for details on programming these controls. The dialog editor is activated by creating a new dialog, clicking a dialog tab at the bottom of the IDE window, or selecting a dialog in the **Macro Organizer** dialog and clicking the **Edit** button.

Initially, a new dialog consists of an empty dialog frame. The next illustration shows Dialog2 of the application Standard library in this state.

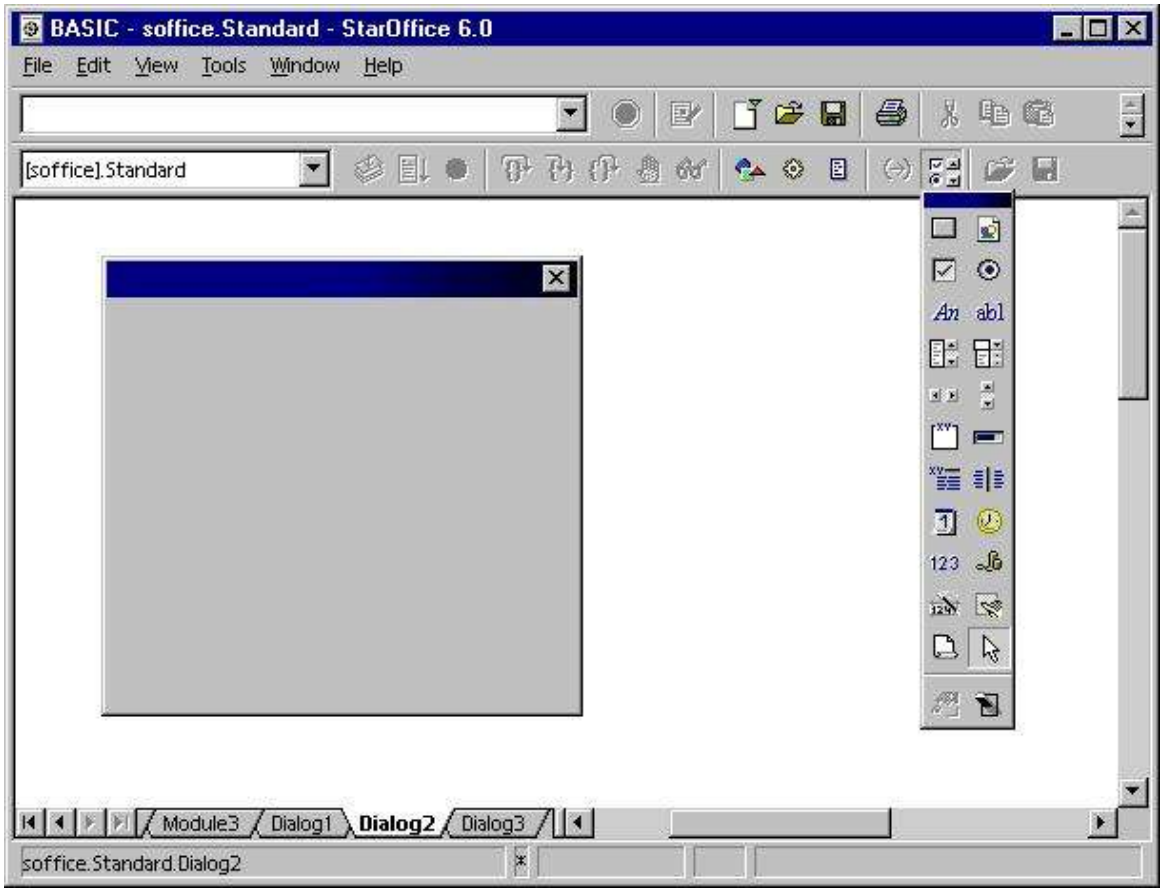


Illustration 161

In the dialog editor mode, the **Controls** button is enabled and the illustration shows the result by clicking this button. A small toolbar with dialog specific tools is displayed. The buttons in this toolbar represent the types of controls that can be inserted into the dialog. The user clicks the desired button, then draws a frame with the mouse at the position to insert the corresponding control type.

The following three buttons in the dialog tools window do not represent controls:

The **Select** button at the lower right of the dialog tools window switches the mouse cursor to selection mode. In this mode, controls are selected by clicking the control with the cursor. If the **Shift** key is held down simultaneously, the selection is extended by each control the user clicks. Controls can also be selected by drawing a rubberband frame with the mouse. All controls that are completely inside the frame will be selected. To select the dialog frame the user clicks its border or includes it in a selection frame completely.

The **Activate Test Mode** button switches on the test mode for dialogs. In this mode, the dialog is displayed as if it was a Basic script (see *11.5 Basic and Dialogs - Programming Dialogs and Dialog Controls*). However, the macros assigned to the controls do not work in this mode. They are thereto help the user design the look and feel of the dialog.

The **Properties** button at the lower left of the dialog tools window opens and closes the **Properties** dialog. This dialog is used to edit all properties of the selected control(s). The next illustration shows the **Properties** dialog for a selected button control.

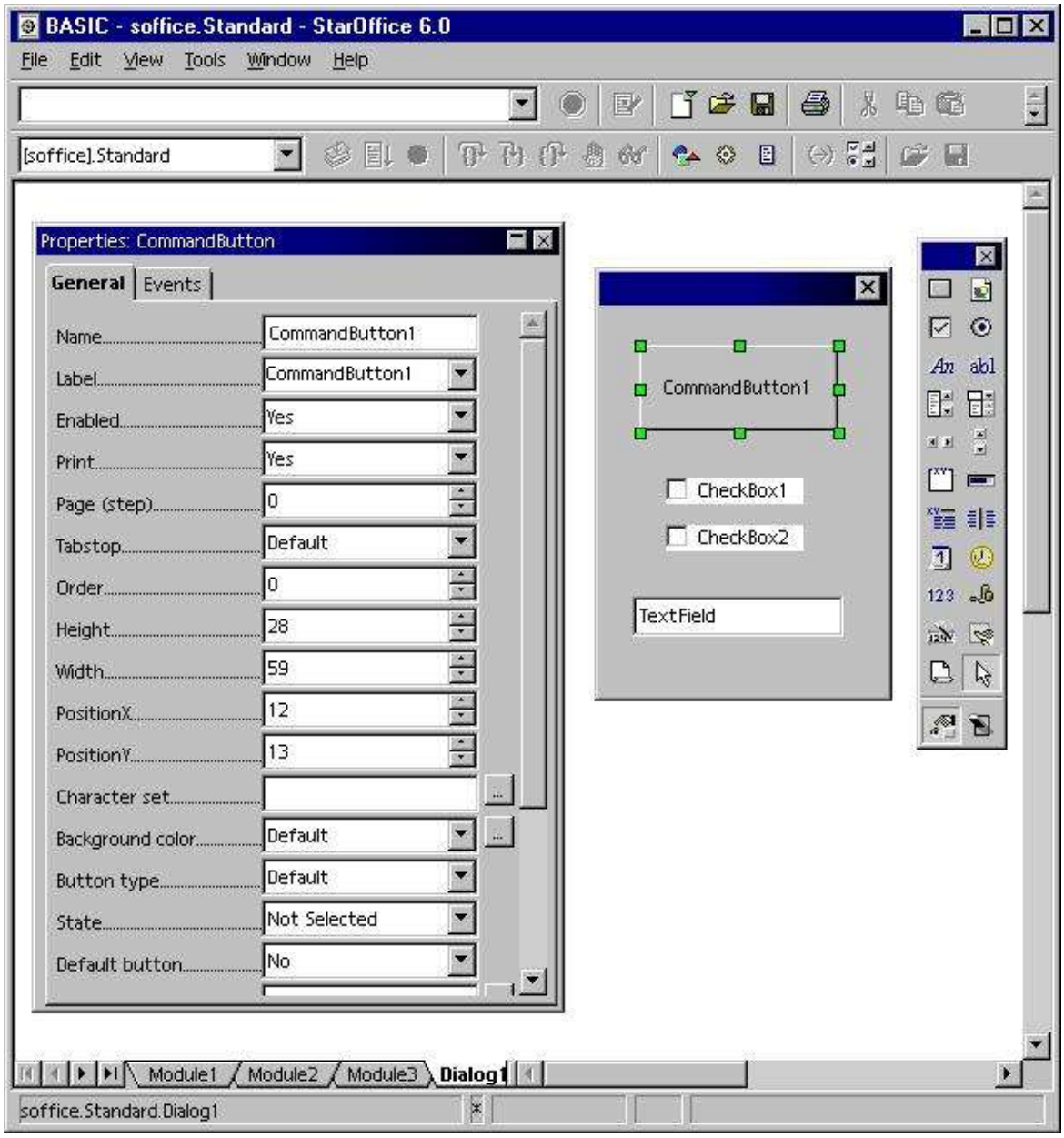


Illustration 162

The illustration above shows that the dialog tool window can be pulled from the main toolbar by dragging the window at its caption bar after opening it.

The **Properties** dialog has two tabs. The **General** tab, visible in Illustration 145, contains a list of properties. Their values are represented by a control. For most properties this is a listbox, such as color and enum types, or an edit field, such as numeric or text properties. For more complex properties, such as fonts or colors, an additional ellipsis button**...** opens another type of dialog, for example, to select a font. When the user changes a property value in an edit field this value is not applied to the control until the edit field has lost the focus. This is forced with the tab key. Alternatively, the user can commit a change by pressing the **Enter** key.

The **Events** tab page displays the macros assigned to the events supported by the selected control:

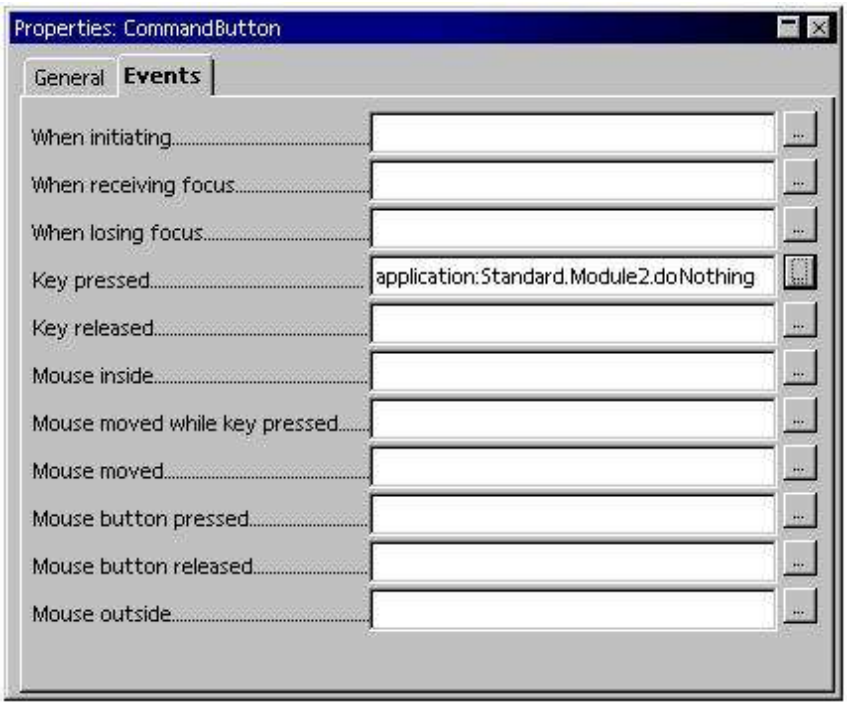


Illustration 163

In the example above, a macro is assigned to the **Key pressed** event: When this event occurs, the displayed Sub `doNothing` in `Module2` of the application Basic library `Standard` is called. The events that are available depend on the type of control selected.

To change the event assignment the user has to click one of the ellipsis**...** buttons to open the **Assign Macro** dialog displayed in Illustration 144.

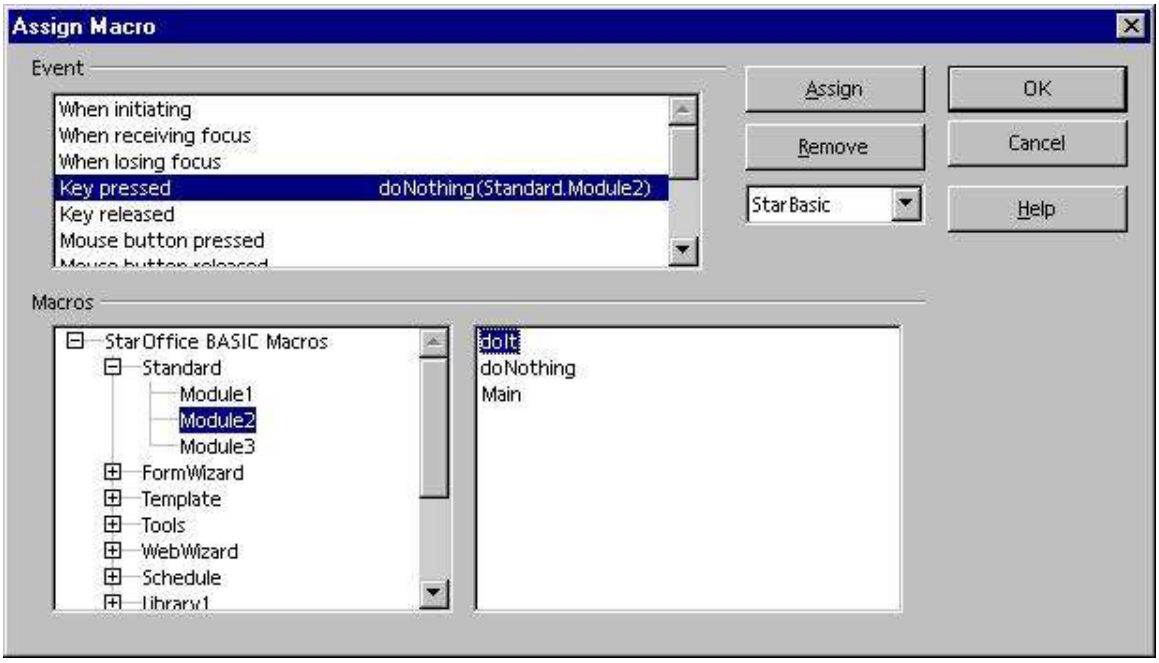


Illustration 164: Assign Macro Dialog

The listbox in the upper left of the dialog titled **Event** displays the same information as the **Events** tab of the **Properties** dialog. The **Assign Macro** dialog is always the same, that is only the selected event in its **Event** list changes according to the ellipsis button the user selected on the **Events** tab of the **Properties** dialog.

To assign a macro to an event, the user selects the event in the **Event** listbox. In the Macro section at the bottom of the dialog, the user navigates through the library hierarchy. If a module is selected in the left listbox, the contained Subs are displayed in the listbox on the right side. The **Assign** button hooks the Sub or Function selected in this listbox to the selected event. If another macro is already assigned to an event, this macro is replaced. If no Sub is selected, the **Assign** button is disabled.

The library hierarchy displayed in the **Macros** listbox contains the application library container and the library container of the document that the edited dialog belongs. If the dialog belongs to an application dialog library, documents are not displayed. Document macros can not be assigned to the controls of application dialogs. Event definition can not depend on a document that is not necessarily loaded when the event occurs.

The **Remove** button is enabled if an event with an assigned macro is selected. Clicking this button removes the macro from the event, therefore the event will have no macro binding.

The listbox below the **Remove** button is used to select different macro languages. Currently, only OpenOffice.org Basic is available.

The **OK** button closes the **Assign Macro** dialog, and applies all event assignments and removals to the control. The changes are reflected on the **Events** tab of the **Properties** dialog.

The **Cancel** button also closes the **Assign Macro**, but all assignment and removal operations are discarded.

As previously explained, it is also possible to select several controls simultaneously. The next picture shows the situation if the user selects both CommandButton1 and CheckBox1.For the **Properties** dialog such a multi selection has some important effects.

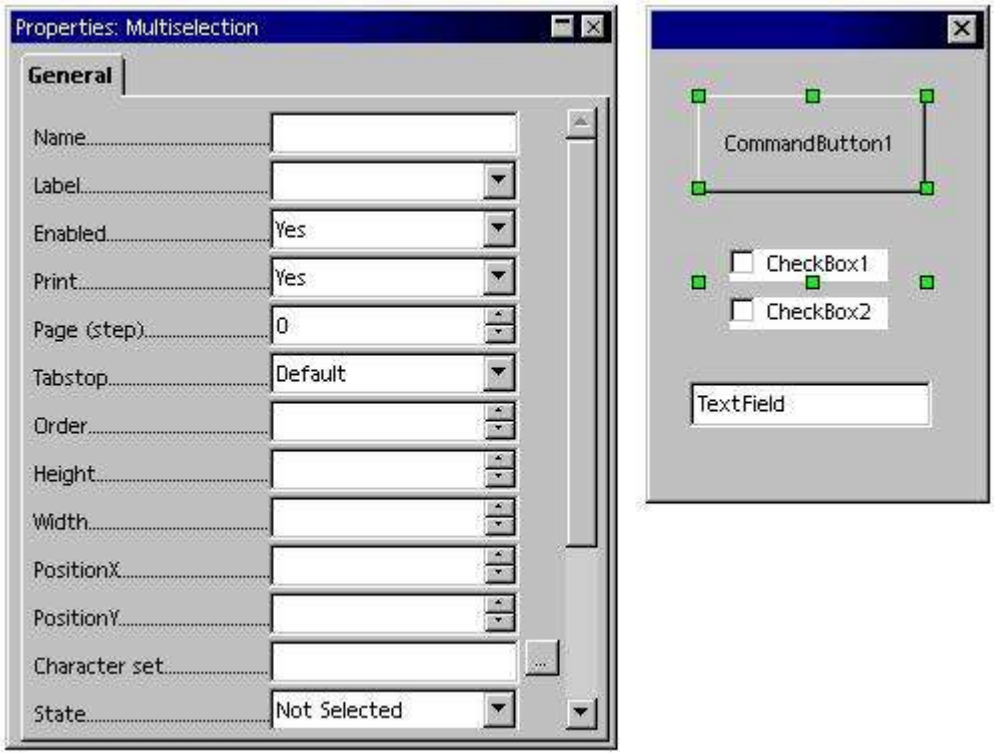


Illustration 165

Here the caption of the **Properties** contains the string Multiselection to point out the special situation. The two important differences compared to the single selection situation are:

- The displayed properties are an intersection of the properties of all the selected controls, that is, the property is only displayed if all the selected controls support that property. A property value is only displayed if the value is the same for all selected controls. All selected controls are effected when a value is changed by the user. Values that are not the same for all controls can be set with the effect that the specified value applies to all controls in the selection.
- A multi-selection **Properties** dialog does not have an **Events** tab. Events can only be specified for single controls.

11.2.3 Assigning Macros to GUI Events

The functionality to assign macros to control events in the dialog editor was discussed earlier. There is also a general functionality to assign macros or other actions to events. This functionality can be accessed through the **Configuration** dialog that is opened using **Tools – Configure...** or by clicking the **Assign...** button in the **Macro** dialog. In this section, only the assignment of macros is discussed. For more information about this dialog, refer to the OpenOffice.org documentation.

The next illustration shows the **Menu** tab of the **Configuration** dialog:

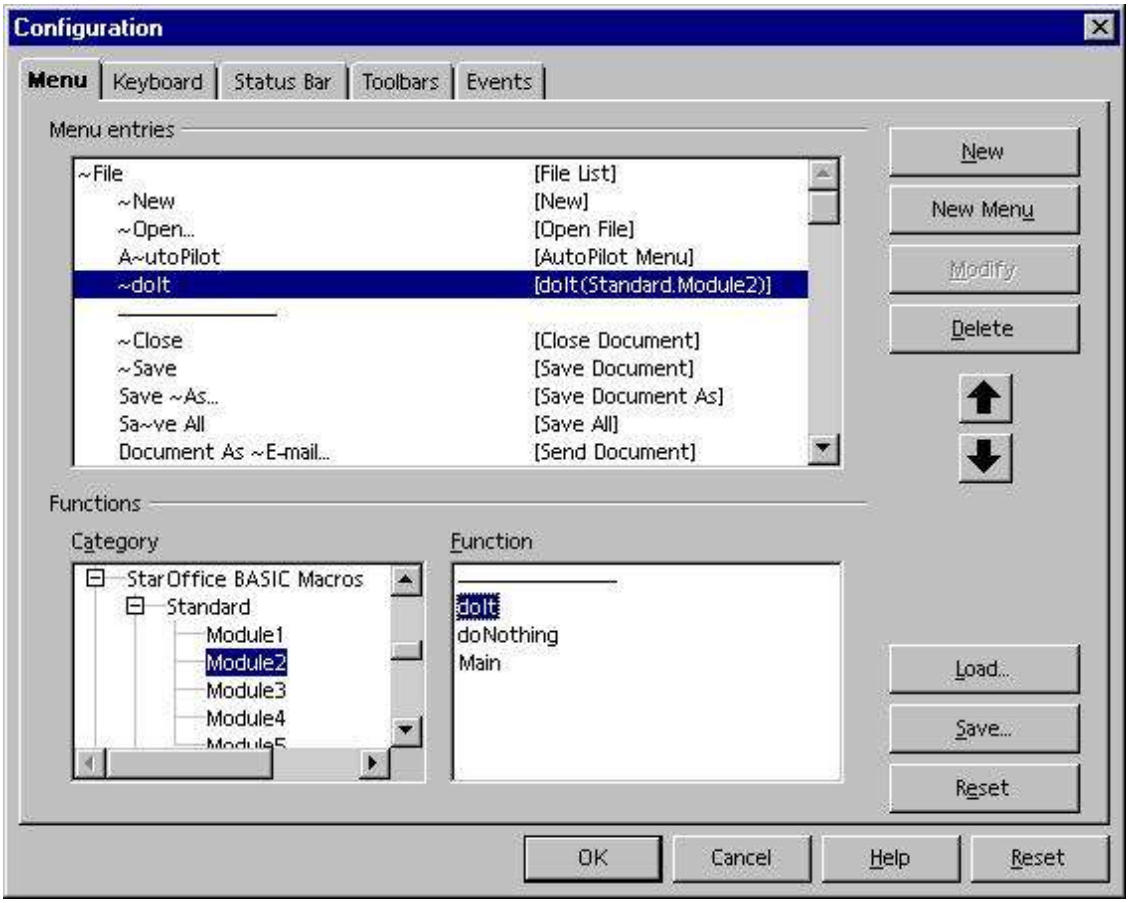


Illustration 166

The illustration above shows how a macro is assigned to a new menu item. The listbox at the top titled **Menu entries** represents the OpenOffice.org menu hierarchy. The **Category** listbox in the lower left corner contains an entry **OpenOffice.org BASIC Macros** that represents the root of the OpenOffice.org application Basic. Other categories not related to macros are also displayed.–. The user navigates through the Basic libraries. When a module is selected, its Subs and Functions are displayed in the **Function** listbox at the lower right.

The **Modify** button assigns the Sub or Function selected in the **Function** listbox to the menu item selected in the **Menu entries** listbox. This button is disabled if a Sub or Function is not selected..

The **New** button creates a new menu item that is connected to the Sub or Function. The other buttons are used for menu design:

- The **Delete** button removes a menu item.
- The arrow buttons change the position of a menu item.
- The **Load... andSave...** buttons load and save a menu configuration.
- The **Reset** button restores the default menu configuration.

The next illustration shows the **Events** tab of the **Configuration** dialog:

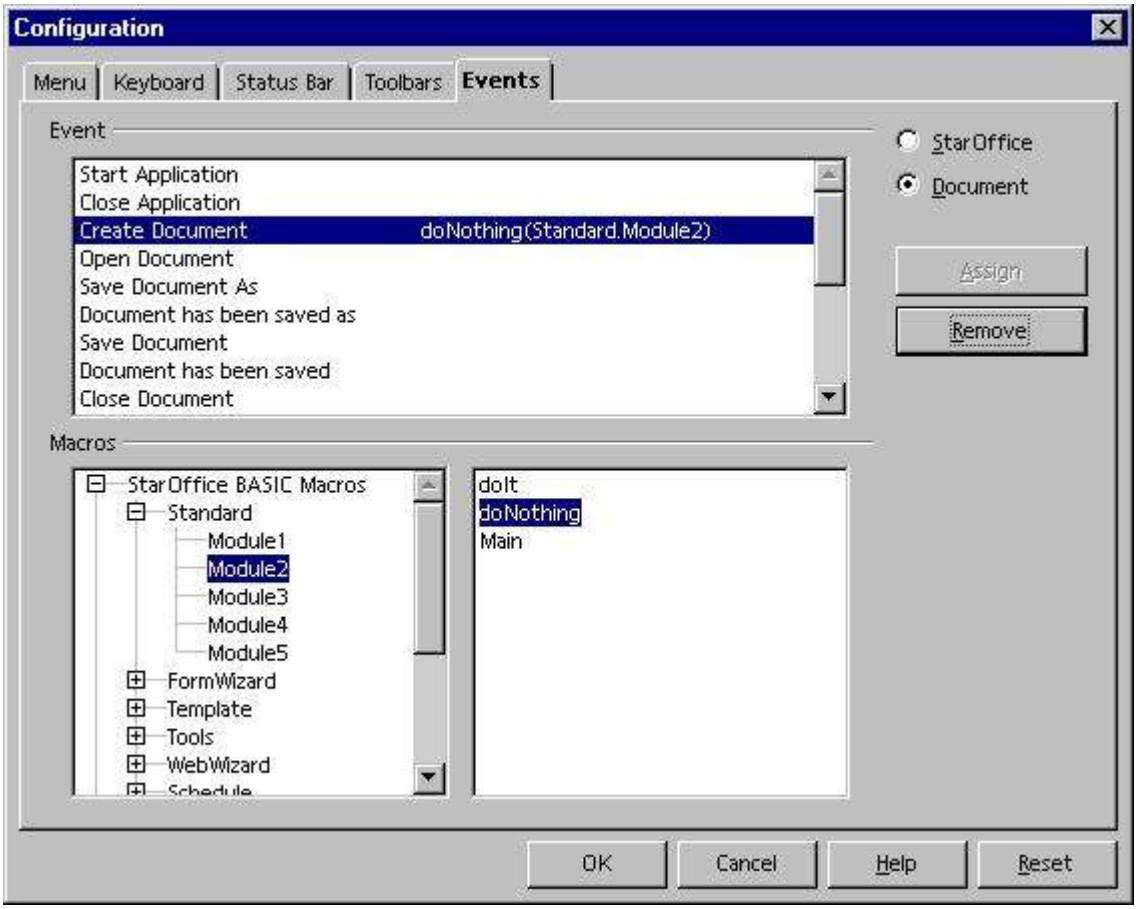


Illustration 167

On this tab, macros can be assigned to general events in OpenOffice.org. The events are listed in the listbox titled **Event**. At the bottom of the dialog the macros can be selected, then be assigned to the selected event with the **Assign** button. This button is disabled if a Sub or Function is not selected. The **Remove** button removes the assigned macro from the selected event. It is disabled if a macro is not assigned to the selected event.

The **OpenOffice.org** radio button is active when the event assignment for the OpenOffice.org application is displayed. This assignment is stored in the OpenOffice.org configuration. The **Document** radio button is active when the event assignment for the current document is displayed. This assignment is made persistent in the document file. If the **Configuration** was not opened from a document, for example, from the Basic IDE, the **Document** radio button is not displayed.

The **Keyboard** tab is similar to the **Menu** and **Events** tabs. Macros are accessed in **Category** and **Function** listboxes, then assigned to a shortcut key that can be specified in the **Shortcut keys** listbox. There are also **Load...**, **Save...** and **Reset** buttons with the same function as the corresponding buttons on the **Menu** tab.

The **Keyboard** tab contains a **OpenOffice.org** and a **Document** radio button with the same functionality as the corresponding radio buttons on the **Events** tab.

11.3 Features of OpenOffice.org Basic

This section provides a general description of the Basic programming language integrated in OpenOffice.org.

11.3.1 Functional Range Overview

This section outlines the functionality provided by OpenOffice.org Basic. The available runtime library functions are also described. The functionality is based upon the Basic online help integrated in OpenOffice.org, but limited to particular functions. Use the Basic online help to obtain further information about the complete Basic functionality.

Apart from the OpenOffice.org API, OpenOffice.org Basic is compatible to Visual Basic.

Screen I/O Functions

Basic provides statements and functions to display information on the screen or to get information from the user:

- The `Print` statement displays strings or numeric expressions in a dialog. Multiple expressions are separated by commas that result in a tab distance between the expressions, or semicolons that result in a space between the expressions. For example:

```
e = 2.718
Print e                  ' displays "2.718"
Print "e =" ; e          ' displays "e = 2.718"
Print "e =" , e          ' displays "e = 2.718"
```

- The `MsgBox` function displays a dialog box containing a message. Additionally, the caption of the dialog, buttons, such as **Ok**, **Cancel**, **Yes** and **No**, and icons, such as question mark and exclamation mark, that are to be displayed are specified. The result then can be evaluated. For example:

```
' display a message box with an exclamation mark and OK and Cancel buttons
ret& = Msgbox ("Changes will be lost. Proceed?", 48 + 1, "Warning")

' show user's selection. 1 = OK, 2 = Cancel
Print ret&
```

- The `InputBox` function displays a prompt in a dialog where the user can input text. The input is assigned to a variable. For example:

```
' display a dialog with "Please enter a phrase:" and "Dear User" as caption
' the dialog contains an edit control and the text entered by the user
' is stored in UserText$ when the dialog is closed with OK. Cancel returns ""
UserText$ = InputBox( "Please enter a phrase:", "Dear User" )
```

File I/O

OpenOffice.org Basic has a complete set of statements and runtime functions to access the operating system's file system that are compatible to Visual Basic. For platform independence, the ability to handle file names in file:// URL notation has been added.

It is not recommended to use this classic Basic file interface in the UNO context, because many interfaces in the OpenOffice.org API expect file I/O specific parameters whose types, for example, `com.sun.star.io.XInputStream` are not compatible to the classic Basic file API.

For programming, the file I/O in OpenOffice.org API context with the service `com.sun.star.ucb.SimpleFileAccess` should be used. This service supports the interface `com.sun.star.ucb.XSimpleFileAccess2`, including the main interface `com.sun.star.ucb.XSimpleFileAccess` that provides fundamental methods to access the file system. The methods are explained in detail in the corresponding interface documentation. The following list provides an overview about the operations supported by this service:

- copy, move and remove files and folders (methods `copy()`, `move()`, `kill()`)
- prompt for information about files and folders (methods `isFolder()`, `isReadOnly()`, `getSize()`, `getContentType()`, `getDateTimeModified()`, `exists()`)
- open or create files (`openFileRead()`, `openFileWrite()`, `openFileReadWrite()`). These functions return objects that support the corresponding stream interfaces `com.sun.star.io.XInputStream`, `com.sun.star.io.XOutputStream` and `com.sun.star.io.XStream`. These interfaces are used to read and write files. The `XSimpleFileAccess2` does not have methods of its own for these operations. Additionally, these interfaces are often necessary as parameters to access methods of several other interfaces. The opened files have to be closed by calling the appropriate methods `com.sun.star.io.XInputStream:closeInput()` or `com.sun.star.io.XOutputStream:closeOutput()`.

  The `XSimpleFileAccess2` also does not have methods to ask for or set the position within a file stream. This is done by calling methods of the `com.sun.star.io.XSeekable` interface that is supported by the objects returned by the `openXXX()` methods.

Two more services are instantiated at the global service manager that extends the service `com.sun.star.ucb.SimpleFileAccess` by functionality specific to text files:

- The service `com.sun.star.io.TextInputStream` supporting `com.sun.star.io.XTextInputStream` and `com.sun.star.io.XActiveDataSink`:

  The service is initialized by passing an object supporting `XInputStream` to the `com.sun.star.io.XActiveDataSink:setInputStream()` method, for example, an object returned by `com.sun.star.ucb.XSimpleFileAccess:openFileRead()`.

  Then the method `com.sun.star.io.TextInputStream:readLine()` and `com.sun.star.io.TextInputStream:readString()` are used to read text from the input stream/file. The method `com.sun.star.io.TextInputStream:isEOF()` is used to check for if the end of the file is reached. The `com.sun.star.io.TextInputStream:setEncoding()` sets a text encoding where UTF-8 is the default.

- The service `com.sun.star.io.TextOutputStream` supporting `com.sun.star.io.XTextOutputStream` and `com.sun.star.io.XActiveDataSource`:

  The service is initialized by passing an object supporting `XOutputStream` to the `com.sun.star.io.XActiveDataSource:setOutputStream()` method, for example, an object returned by `com.sun.star.ucb.XSimpleFileAccess:openFileWrite()`.

  Then the method `com.sun.star.io.XTextOutputStream:writeString()` is used to write text to the output stream.

Date and Time Functions

OpenOffice.org Basic supports several Visual Basic compatible statements and functions to perform date and time calculations. The functions are `DateSerial`, `DateValue`, `Day`, `Month`, `WeekDay`, `Year`, `Hour`, `Now`, `Second`, `TimeSerial`, `TimeValue`, `Date`, `Time`, and `Timer`.

The function `Date` returns the current system date as a string and the function `Time` returns the current system time as a string. The other functions are not explained.

In the UNO/toolkit controls context there are two other functions. The date field control method `com.sun.star.awt.XDateField:setDate()` expects the date to be passed as a long value in a special ISO format and the `com.sun.star.awt.XDateField:getDate()` returns the date in this format.

The Basic runtime function `CDateToIso` converts a date from the internal Basic date format to the required ISO date format. Since the string date format returned by the `Date` function is converted to the internal Basic date format automatically, `Date` can be used directly as an input parameter for `CDateToIso`:

```
IsoDate = CDateToIso(Date)
oTextField.setDate(IsoDate)
```

The runtime function `CDateToIso` represents the reverse operation and converts a date from the ISO date format to the internal Basic date format.

```
Dim aDate as date
aDate = CDateFromIso(IsoDate)
```

Please see also *11.5 Basic and Dialogs - Programming Dialogs and Dialog Controls* in this context.

Numeric Functions

OpenOffice.org Basic supports standard numeric functions, such as:

- `Cos` calculating the cosine of an angle
- `Sin` calculating the sine of an angle
- `Tan` calculating the tangent of an angle
- `Atn` calculating the arctangent of a numeric value
- `Exp` calculating the base of the natural logarithm (e = 2.718282) raised to a power
- `Log` calculating the natural logarithm of a number
- `Sqr` calculating the square root of a numeric value
- `Abs` calculating the absolute value of a numeric value
- `Sgn` returning -1 if the passed numeric value is negative, 1 if it is positive, 0 if it is zero.

String Functions

OpenOffice.org Basic supports several runtime functions for string manipulation. Some of the functions are explained briefly in the following:

- `Asc` returns the the Unicode value.
- `Chr` returns a string containing the character that is specified by the ASCII or Unicode value passed to the function. This function is used to represent characters, such as "" or the carriage return code (`chr(13)`) that can not be written in the "" notation.
- `Str` converts a numeric expression to a string.
- `Val` converts a string to a numeric value.

- `LCase` converts all letters in a string to lowercase. Only uppercase letters within the string are converted. All lowercase letters and nonletter characters remain unchanged.
- `UCase` converts characters in a string to uppercase. Only lowercase letters in a string are affected. Uppercase letters and all other characters remain unchanged.
- `Left` returns the leftmost “n” characters of a string expression.
- `Mid` returns the specified portion of a string expression.
- `Right` returns the rightmost "n" characters of a string expression.
- `Trim` removes all leading and trailing spaces of a string expression.

Specific UNO Functions

The UNO specific runtime functions `CreateUnoListener`, `CreateUnoService`, `GetProcessServiceManager`, `HasUnoInterfaces`, `IsUnoStruct`, `EqualUnoObjects` are described in *3.4.3 Professional UNO - UNO Language Bindings - OpenOffice.org Basic*.

11.3.2 Accessing the UNO API

In *3.4.3 Professional UNO - UNO Language Bindings - OpenOffice.org Basic*, the interaction between Basic and UNO is described on an elementary level. This section describes the interface between Basic and the UNO API at the level of the OpenOffice.org application.

This is realized by two predefined Basic properties:

- `StarDesktop`
- `ThisComponent`

The property `StarDesktop` gives access to the global OpenOffice.org application API while the property `ThisComponent` accesses the document related API.

StarDesktop

The property `StarDesktop` is a shortcut for the service com.sun.star.frame.Desktop.

Example:

```
MsgBox StarDesktop.Dbg_SupportedInterfaces

' is the same as

Dim oDesktop
oDesktop = CreateUnoService( "com.sun.star.frame.Desktop" )
MsgBox oDesktop.Dbg_SupportedInterfaces
```

The displayed message box differs slightly because `Dbg_SupportedInterfaces` displays "StarDesktop" as an object type of the desktop object in the first case and "com.sun.star.frame.Desktop" in the second. But the two objects are the same.

ThisComponent

The property `ThisComponent` is used from document Basic, where it represents the document the Basic belongs to. The type of object accessed by `ThisComponent` depends on the document type. The following example shows the differences.

Basic module in a OpenOffice.org document:

```
Sub Main
    MsgBox ThisComponent.Dbg_SupportedInterfaces
End Sub
```

The execution of this Basic routine shows different results for a Text, Spreadsheet and Presentation document. Depending on the document type, a different set of interfaces are supported by the object. A portion of the interfaces are common to all these document types representing the general functionality that documents of any type offer. In particular, all OpenOffice.org documents support the `com.sun.star.document.OfficeDocument` service, including the interfaces `com.sun.star.frame.XStorable` and `com.sun.star.view.XPrintable`. Another interface is `com.sun.star.frame.XModel`.

The following list shows the interfaces supported by all document types:

```
com.sun.star.beans.XPropertySet
com.sun.star.container.XChild
com.sun.star.document.XDocumentInfoSupplier
com.sun.star.document.XEventBroadcaster
com.sun.star.document.XViewDataSupplier
com.sun.star.document.XEventsSupplier
com.sun.star.document.XLinkTargetSupplier
com.sun.star.frame.XModel
com.sun.star.frame.XStorable
com.sun.star.lang.XServiceInfo
com.sun.star.lang.XMultiServiceFactory
com.sun.star.lang.XEventListener
com.sun.star.style.XStyleFamiliesSupplier
com.sun.star.util.XModifiable
com.sun.star.view.XPrintable
```

For more information about the functionality of these interfaces, see *6.1.1 Office Development - OpenOffice.org Application Environment - Overview - Framework API - Frame-Controller-Model Paradigm*. This section alsogoes into detail about the general document API.

In addition to the common services or interfaces, each document type supports specific services or interfaces. The following list outlines the supported services and important interfaces:

A Text document supports:

- The service `com.sun.star.text.TextDocument` supports the interface `com.sun.star.text.XTextDocument`.
- Several interfaces, especially from the `com.sun.star.text` package.

A Spreadsheet document supports:

- The service `com.sun.star.sheet.SpreadsheetDocument`,
- The service `com.sun.star.sheet.SpreadsheetDocumentSettings`.
- Several other interfaces, especially from the `com.sun.star.sheet` package.

Presentation and Drawing documents support:

- The service `com.sun.star.drawing.DrawingDocument`.
- Several other interfaces, especially from the `com.sun.star.drawing` package.

The usage of these services and interfaces is explained in the document type specific chapters *7 Text Documents*, *8 Spreadsheet Documents* and *9 Drawing*.

As previously mentioned, `ThisComponent` is used from document Basic, but it is also possible to use it from application Basic. In an application wide Bais module, `ThisComponent` is identical to the current component that can also be accessed through `StarDesktop.CurrentComponent`. The

only difference between the two is that if the BasicIDE is active, `StarDesktop.CurrentComponent` refers to the BasicIDE itself while `ThisComponent` always refers to the component that was active before the BasicIDE became the top window.

11.3.3 Special Behavior of OpenOffice.org Basic

Threading and rescheduling of OpenOffice.org Basic differs from other languages which must be taken into consideration.

Threads

OpenOffice.org Basic does not support threads:

- In situations it may be necessary to create new threads to access UNO components in a special way. This is not possible in OpenOffice.org Basic.
- OpenOffice.org Basic is unable to control threads. If two threads use the Basic runtime system simultaneously, the result will be undefined results or even a crash. Please take precautions.

Rescheduling

The OpenOffice.org Basic runtime system reschedules regularly. It allows system messages to be dispatched continuously that have been sent to the OpenOffice.org process during the runtime of a Basic module. This is necessary to allow repainting operations, and access to controls and menus during the runtime of a Basic script as Basic runs in the OpenOffice.org main thread. Otherwise, it would not be possible to stop a running Basic script by clicking the corresponding button on the toolbar.

This behavior has an important consequence. Any system message, for example, clicking a push button control, can result in a callback into Basic if an corresponding event is specified. The Basic programmer must be aware of the fact that this can take place at any point of time when a script is running.

The following example shows how this effects the state of the Basic runtime system:

```
Dim EndLoop As Boolean
Dim AllowBreak As Boolean

' Main sub, the execution starts here
Sub Main
    ' Initialize flags
    EndLoop = FALSE
    AllowBreak = FALSE

    Macro1   ' calls sub Macro1
End Sub


' Sub called by main
Sub Macro1
    Dim a
    While Not EndLoop
        ' Toggle flags permanently
        AllowBreak = TRUE
        AllowBreak = FALSE
    Wend
    Print "Ready!"
End Sub

' Sub assigned to a bush button in a writer document
Sub Break
If AllowBreak = TRUE Then
```

```
        EndLoop = TRUE
        EndIf
    End Sub
```

When `Sub Main` in this Basic module is executed, the two Boolean variables `EndLoop` and `AllowBreak` are initialized. Then `Sub Macro1` is called where the execution runs into a loop. The loop is executed until the `EndLoop` flag is set to `TRUE`. This is done in `Sub Break` that is assigned to a push button in a writer document, but the `EndLoop` flag can only be set to `TRUE` if the `AllowBreak` flag is also `TRUE`. This flag is permanently toggled in the loop in `Sub Macro1`.

The program execution may or may not be stopped if the push button is clicked. It depends on the point of time the push button is clicked. If the Basic runtime system has just executed the `AllowBreak = TRUE` statement, the execution is stopped because the `If` condition in `Sub Break` is `TRUE` and the `EndLoop` flag can be set to `TRUE`. If the push button is clicked when the `AllowBreak` variable is `FALSE`, the execution is not stopped. The Basic runtime system reschedules permanently, therefore it is unpredictable. This is an example to show what problems may result from the Basic rescheduling mechanism.

Callbacks to Basic that result from rescheduling have the same effect as if the Sub specified in the event had been called directly from the position in the Basic code that is executed in the moment the rescheduling action leading to the callback takes place. In this example, the Basic call stack looks like this if a breakpoint is placed in the `Sub Break`:

```
Basic                Native code

0: Break  <---   Callback due to push button event
1: Macro1  --->  Reschedule()
2: Main
```

With the call to the native `Reschedule` method, the Basic runtime system is left and reentered when the push button events in a Callback to Basic. On the Basic stack this looks like a direct call from `Sub Macro1` to `Sub Break`.

A similar situation occurs when a program raises a dialog using the execute method of the dialog object returned by `CreateUnoDialog()`. See *11.5 Basic and Dialogs - Programming Dialogs and Dialog Controls*. In this case, the Basic runtime system does not reschedule, but messages are processed by the dialog's message loop that also result in callbacks to Basic. When the Basic runtime system is called back due to an event at a dialog control, the resulting Basic stack looks analogous. For example:

```
Sub Main
    Dim oDialog
    oDialog = CreateUnoDialog( ... )
    oDialog.execute()
End Sub

Sub DoIt
        ...
End Sub
```

If `Sub Doit` is specified to be executed if an event occurs for one of the dialog controls, the Basic call stack looks like this if a breakpoint is placed in `Sub DoIt`:

```
Basic                Native code

0: DoIt   <---   Callback due to control event
1: Main    --->  execute() ---> Reschedule()
```

There is also a difference to the rescheduling done directly by the Basic runtime system. The rescheduling done by the dialog's message loop can not result in unpredictable behavior, because the Basic runtime system has called the dialog's `execute` method and waits for its return. It is in a well-defined state.

11.4 Advanced Library Organization

Basic source code and Dialogs are organized in libraries. This section describes the structure and usage of the library system.

11.4.1 General Structure

The library system that is used to store Basic source code modules and Dialogs has three levels:

Library container
: The library container represents the top level of the library hierarchy containing libraries. The libraries inside a library container are accessed by name.

Library
: A library contains elements that logically belong together, for example, several Basic modules that form a program or a set of related dialogs together.

Library elements
: Library elements are Basic source code modules or dialogs. The elements represent the lowest level in the library hierarchy. For Basic source code modules, the element type is string. Dialogs are represented by the interface `com.sun.star.io.XInputStreamProvider` that provides access to the XML data describing the dialog.

The hierarchy is separated for Basic source code and dialogs, that is, a Basic library container only contains Basic libraries containing Basic source code modules and a dialog library container only contains dialog libraries containing dialogs.

Basic source code and dialogs are stored globally for the whole office application and locally in documents. For the application, there is one Basic library container and one dialog library container. Every document has one Basic library container and one dialog library container as well. By including the application or document level, the library system actually has four levels. Illustration 143: Basic source editor window depicts this structure.

As shown in the library hierarchy for Document 1, the Basic and dialog library containers do not have the same structure. The Basic library container has a library named Library1 and the dialog library container has a library named Library2. The library containers are separated for Basic and dialogs in the API.

It is not recommended to create a structure as described above because the library and dialog containers are not separated in the GUI, for example, in the **Tools - Macro...** dialog. When a user creates or deletes a new library through **Tools - Macro – Organizer...,** the library is created or deleted in the Basic and the dialog library containers.

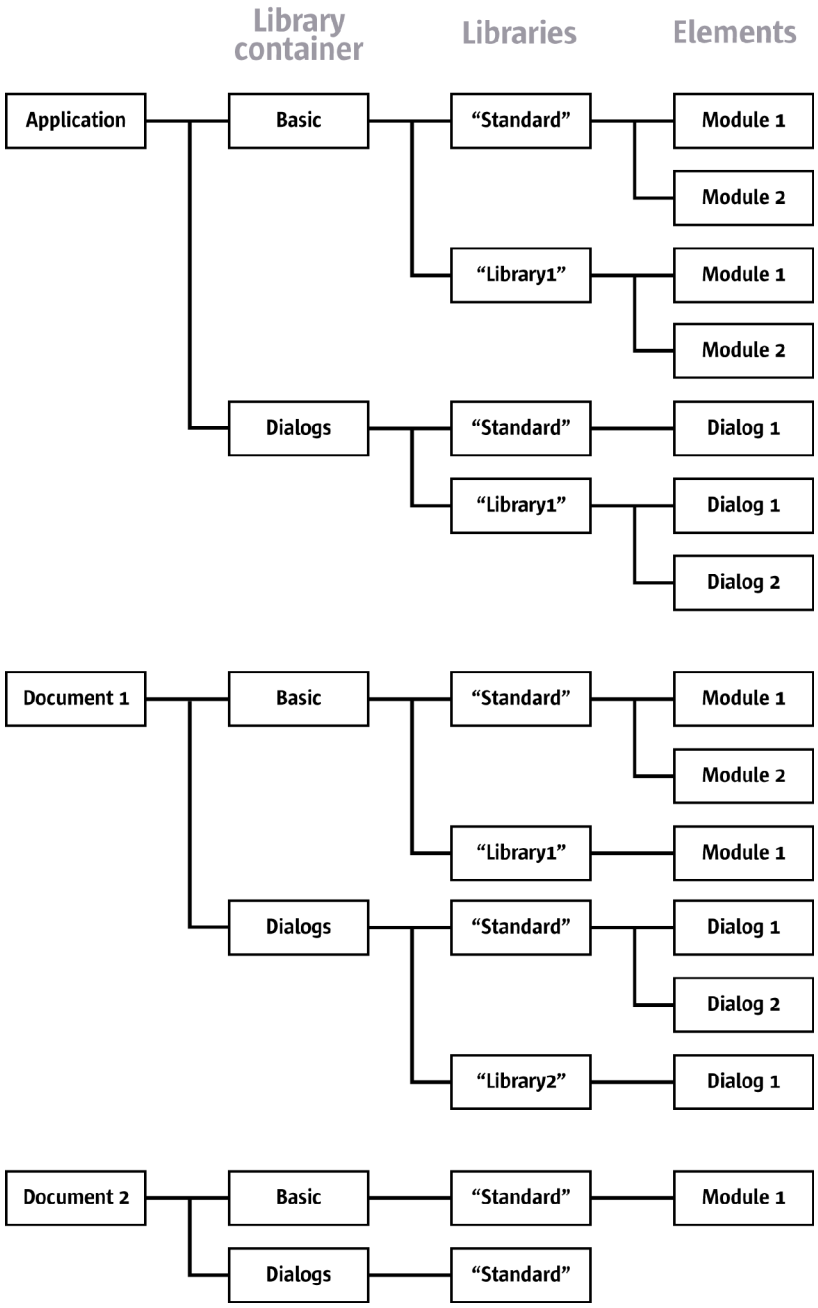


Illustration 168: Sample module structure

11.4.2 Accessing Libraries from Basic

Library Container Properties in Basic

Currently, the library system is implemented using UNO interfaces, not as a UNO service. Therefore, the library system cannot be accessed by instantiating an UNO service. The library system has to be accessed directly from Basic using the built-in properties `BasicLibraries` and `DialogLibraries`.

The `BasicLibraries` property refers to the Basic library container that belongs to the library container that the `BasicLibraries` property is accessed. In an *application*-wide Basic module, the property `BasicLibraries` accesses the *application* Basic library containerand in a *document* Basic module, the property `BasicLibraries` contains the *document* Basic library container. The same applies to the `DialogLibraries` property.

Loading Libraries

Initially, most Basic libraries are not loaded. All the libraries in the application library container are known after starting OpenOffice.org, and all the library elements in a document are known when it is loaded, most of them are *disabled* until they are loaded explicitly. This mechanism saves time during the Basic initialization. When a Basic library is initialized, the source code modules are inserted into the Basic engine and compiled. If there are many libraries with big modules, it is tim consuming, especially if the libraries are not required.

The exception to this is that every library container contains a library named "Standard" that is always loaded. This library is used as a standard location for Basic programs and dialogs that do not need a complex structure. All other libraries have to be loaded explicitly. For example:

When Library1, Module1 looks like

```
Sub doSomething
    MsgBox "doSomething"
End Sub
```

the following code in library Standard, Module1

```
Sub Main
    doSomething()
End Sub
```

fails, unless the user loaded Library1 before using the **Tools - Macro** dialog. A runtime error "Property or method not found" occurs. To avoid this, load library Library1 before calling `doSomething()`:

```
Sub Main
    BasicLibraries.loadLibrary( "Library1" )
    doSomething()
End Sub
```

Accordingly in the dialog container, all the libraries besides the Standard library have to be loaded before the dialogs inside the library can be accessed. For example:

```
Sub Main
    ' If this line was missing the following code would fail
    DialogLibraries.loadLibrary( "Library1" )

    ' Code to instantiate and display a dialog
    ' Details will be explained in a later chapter
    oDlg = createUnoDialog( DialogLibraries.Library1.Dialog1 )
    oDlg.execute()
End Sub
```

The code to instantiate and display the dialog is described in *11.5 Basic and Dialogs - Programming Dialogs and Dialog Controls*. The library representing `DialogLibraries.Library1.Dialog1` is only valid once `Library1` has been loaded.

The properties `BasicLibraries` and `DialogLibraries` refer to the container that includes the Basic source accessing these properties. Therefore in a document module Basic the properties `BasicLibraries` and `DialogLibraries` refer to the Basic and Dialog library container of the document. In most cases, libraries in the document have to be loaded. In other cases it might be necessary to access application-wide libraries from document Basic. This can be done using the `GlobalScope` property. The `GlobalScope` property represents the root scope of the application Basic, therefore the application library containers can be accessed as properties of `GlobalScope`.

Example module in a Document Basic in library Standard:

```
Sub Main
   ' This code loads Library1 of the
...' Document Basic library container
   BasicLibraries.loadLibrary( "Library1" )

   ' This code loads Library1 of the
...' Document dialog library container
   DialogLibraries.loadLibrary( "Library1" )

   ' This code loads Library1 of the
...' Application Basic library container
   GlobalScope.BasicLibraries.loadLibrary( "Library1" )

   ' This code loads Library1 of the
...'  Application  dialog library container
   GlobalScope.DialogLibraries.loadLibrary( "Library1" )

   ' This code displays the source code of the
...' Application Basic module Library1/Module1
   MsgBox GlobalScope.BasicLibraries.Library1.Module1
End Sub
```



Application library containers can be accessed from document-embedded Basic libraries using the `GlobalScope` property, for example, `GlobalScope.BasicLibraries.Library1`.

Library Container API

The `BasicLibraries` and `DialogLibraries` support `com.sun.star.script.XLibraryContainer2` that inherits from `com.sun.star.script.XLibraryContainer`, which is a `com.sun.star.container.XNameContainer`. Basic developers do not require the location of the interface to use a method, but a basic understanding is helpful when looking up the methods in the API reference.

The `XLibraryContainer2` handles existing library links and the write protection for libraries. It is also used to rename libraries:

```
boolean isLibraryLink( [in] string Name)
string getLibraryLinkURL( [in] string Name)
boolean isLibraryReadOnly( [in] string Name)
void setLibraryReadOnly( [in] string Name,
                         [in] boolean bReadOnly)
void renameLibrary( [in] string Name, [in] string NewName)
```

The `XLibraryContainer` creates and removes libraries and library links. Furthermore, it can test if a library has been loaded or, if necessary, load it.

```
com::sun::star::script::XNameContainer createLibrary( [in] string Name)
com::sun::star::script::XNameAccess createLibraryLink( [in] string Name,
                                        [in] string StorageURL, [in] boolean ReadOnly)
void removeLibrary( [in] string Name)
boolean isLibraryLoaded( [in] string Name)
void loadLibrary( [in] string Name)
```

The methods of `XNameContainer` access and manage the libraries in the container:

```
void insertByName( [in] string name, [in] any element)
void removeByName( [in] string name)
any getByName( [in] string name)

void replaceByName( [in] string name, [in] any element)

sequence < string > getElementNames()
boolean hasByName( [in] string name)
type getElementType()
boolean hasElements()
```

These methods are accessed using the UNO API as described in *3.4.3 Professional UNO - UNO Language Bindings - OpenOffice.org Basic*. Note however, these interfaces can only be used from OpenOffice.org Basic, not from other environments.

Libraries can be added to library containers in two different ways:

Creating a New Library

Creating a new library is done using the `createLibrary()` method. A library created with this method belongs to the library container where `createLibrary()` has been called. The implementation of the library container is responsible for saving and loading this library. This functionality is not currently covered by the interfaces, therefore the implementation determines how and where this is done. The method `createLibrary()` returns a standard `com.sun.star.container.XNameContainer` interface to access the library elements and modify the library.

Initially, such a library is empty and new library elements are inserted. It is also possible to protect a library from changes using the `setLibraryReadOnly()` method. In a read-only library, no elements can be inserted or removed, and the modules or dialogs inside cannot be modified in the BasicIDE. For additional information, see *11.2 Basic and Dialogs - OpenOffice.org Basic IDE*. Currently, the read-only status can only be changed through API.

Creating a Link to an Existing Library

Creating a link to an existing library is accomplished using the method `createLibraryLink()`. Its `StorageURL` parameter describes the location where the library *.xlb* file is stored. For additional information about this topic, see the section on *11.7 Basic and Dialogs - Library File Structure*). A library link is only referenced by the library container and is not owned, therefore the library container is not responsible for the location to store the library. This is only described by the `StorageURL` parameter.

The `ReadOnly` parameter sets the read-only status of the library link. This status is independent of the read-only status of the linked library. A linked library is only modified when the library and link to the library are *not* read only. For example, this mechanism provides read-only access to a library located on a network drive without forcing the library to be read-only, thus the library can be modified easily by an authorized person without changing its read-only status.

The following tables provides a brief overview about other methods supported by the library containers:

Selected Methods of `com.sun.star.script.XLibraryContainer2`	
`isLibraryLink()`	`boolean`. Can be used to ask if a library was added to the library container as a link.
`getLibraryLinkURL()`	`string`. Returns the StorageURL for a linked library. This corresponds to the StorageURL parameter of the `createLibraryLink(...)` method and is primarily meant to be displayed to the users through the graphical user interface.
`isLibraryReadOnly()`	`boolean`. Retrieves the read-only status of a library. In case of a library link, the method returns only `false, that is`, the library can be modified, when the link or the linked library are not read only.
`renameLibrary()`	Assigns a new name to a library. If the library was added to the library container as a link, only the link is renamed.

Selected Methods of `com.sun.star.script.XLibraryContainer`	
`loadLibrary()`	`void`. Loads a library. This is explained in detail in section *11.4 Basic and Dialogs - Advanced Library Organization*
`isLibraryLoaded()`	`boolean`. Allows the user to find out if a library has already been loaded.

Selected Methods of com.sun.star.script.XLibraryContainer	
removeLibrary()	void. Removes the library from the library container. If the library was added to the library container as a link, only the link is removed, because the library addressed by the link is not considered to be owned by the library container.

11.4.3 Variable Scopes

Some aspects of scoping in Basic depend on the library structure. This section describes which variables declared in a Basic source code module are seen from what libraries or modules. Generally, only variables declared outside Subs are affected by this issue. Variables declared inside Subs are local to the Sub and not accessible from outside of the Sub. For example:

```
Option Explicit      ' Forces declaration of variables

Sub Main
    Dim a%
    a% = 42  ' Ok
    NotMain()
End Sub

Sub NotMain
        a% = 42       ' Runtime Error "Variable not defined"
End Sub
```

Variables can also be declared outside of Subs. Then their scope includes at least the module they are declared in. To declare variables outside of the Subs, the commands `Private`, `Public/Dim` and `Global` are used.

The `Private` command is used to declare variables that can only be used locally in a module. If the same variable is declared as `Private` in two different modules, they are used independently in each module. For example:

Library Standard, Module1:

```
Private x As Double

Sub Main
    x = 47.11                ' Initialize x of Module1
    Module2_InitX            ' Initialize x of Module2

    MsgBox x                 ' Displays the x of Module1
    Module2_ShowX            ' Displays the x of Module2
End Sub
```

Library Standard, Module2:

```
Private x As Double

Sub Module2_InitX
    x = 47.12                ' Initialize x of Module2
End Sub

Sub Module2_ShowX
    MsgBox x                 ' Displays the x of Module2
End Sub
```

When `Main` in Module1 is executed, `47.11` is displayed (`x` of Module1) and then `47.12` (`x` of Module2).

The `Public and Dim` commands declare variables that can also be accessed from outside the module. They are identical in this context. Variables declared with `Public` and `Dim` can be accessed from all modules that belong to the same library container. For example, based on the library structure shown in Illustration 143: Basic source editor window, any variable declared with `Public` and `Dim` in the Application Basic Modules Standard/Module1, Standard/Module2, Library1/Module1, Library1/Module2 can also be accessed from all of these modules, therefore the library container represents the logical root scope.

11.5 Programming Dialogs and Dialog Controls

The dialogs and dialog controls are UNO components that provide a graphical user interface belonging to the module `[MOUDLE:com.sun.star.awt]`. The Toolkit controls follow the Model-View-Controller (MVC) paradigm, which separates the component into three logical units, the *model*, *view,* and *controller*. The model represents the data and the low-level behavior of the component. It has no specific knowledge of its controllers or its views. The view manages the visual display of the state represented by the model. The controller manages the user interaction with the model.



Note, that the Toolkit controls combine the view and the controller into one logical unit, which forms the user interface for the component.

The following example of a text field illustrates the separation into model, view and controller. The model contains the data which describes the text field, for example, the text to be displayed, text color and maximum text length. The text field model is implemented by the `com.sun.star.awt.UnoControlEditModel` service that extends the `com.sun.star.awt.UnoControlModel` service. All aspects of the model are described as a set of properties which are accessible through the `com.sun.star.beans.XPropertySet` interface. The view is responsible for the display of the text field and its content. It is possible to have multiple views for the same model, but not for Toolkit dialogs. The view is notified about model changes, for example, changes to the text color property causes the text field to be repainted. The controller handles the user input provided through thekeyboard and mouse. If the user changes the text in the text field, the controller updates the corresponding model property. In addition, the controller updates the view, for example, if the user presses the delete button on the keyboard, the marked text in the text field is deleted. A more detailed description of the MVC paradigm can be found in the chapter about forms *13 Forms*.

The base for all the Toolkit controls is the `com.sun.star.awt.UnoControl` service that exports the following interfaces:

- The `com.sun.star.awt.XControl` interface specifies control basics.For example, it gives access to the model, view and context of a control.
- The `com.sun.star.awt.XWindow` interface specifies operations for a window component.
- The `com.sun.star.awt.XView` interface provides methods for attaching an output device and drawing an object.

11.5.1 Dialog Handling

Showing a Dialog

After a dialog has been designed using the dialog editor, a developer wants to show the dialog from within the program code. The necessary steps are shown in the following example: (BasicAndDialogs/ToolkitControls)

```
Sub ShowDialog()

    Dim oLibContainer As Object, oLib As Object
    Dim oInputStreamProvider As Object
    Dim oDialog As Object

    Const sLibName = "Library1"
    Const sDialogName = "Dialog1"

    REM library container
```

```
    oLibContainer = DialogLibraries

    REM load the library
    oLibContainer.loadLibrary( sLibName )

    REM get library
    oLib = oLibContainer.getByName( sLibName )

    REM get input stream provider
    oInputStreamProvider = oLib.getByName( sDialogName )

    REM create dialog control
    oDialog = CreateUnoDialog( oInputStreamProvider )

    REM show the dialog
    oDialog.execute()

End Sub
```

The dialog control is created by calling the runtime function `CreateUnoDialog()` which takes an object as parameter that supports the `com.sun.star.io.XInputStreamProvider` interface. This object provides an input stream that represents an XML description of the dialog. The section *11.4 Basic and Dialogs - Advanced Library Organization* explains the accessing to the object inside the library hierarchy. The dialog control is shown by calling the `execute()` method of the `com.sun.star.awt.XDialog` interface. It can be closed by calling `endExecute()`, or by offering a **Cancel** or **OK** Button on the dialog. For additional information, see *11.5 Basic and Dialogs - Programming Dialogs and Dialog Controls*.

Getting the Dialog Model

If a developer wants to modify any properties of a dialog or a control, it is necessary to have access to the dialog model. From a dialog, the model can be obtained by the `getModel` method of the `com.sun.star.awt.XControl` interface

```
oDialogModel = oDialog.getModel()
```

or shorter

```
oDialogModel = oDialog.Model
```

Dialog as Control Container

All controls belonging to a dialog are grouped together logically. This hierarchy concept is reflected by the fact that a dialog control is a container for other controls. The corresponding service `com.sun.star.awt.UnoControlDialog` therefore supports the `com.sun.star.awt.XControlContainer` interface thatoffers container functionality, namely access to its elements by name. Since in OpenOffice.org Basic, every method of every supported interface is called directly at the object without querying for the appropriate interface, a control with the name TextField1 can be obtained from a dialog object `oDialog` simply by:

```
oControl = oDialog.getControl("TextField1")
```

See *3.4.3 Professional UNO - UNO Language Bindings - OpenOffice.org Basic* for additional information. The hierarchy between a dialog and its controls can be seen in the dialog model `com.sun.star.awt.UnoControlDialogModel`, which is a container for control models and therefore supports the `com.sun.star.container.XNameContainer` interface. A control model is obtained from a dialog model by:

```
oDialogModel = oDialog.getModel()
oControlModel = oDialogModel.getByName("TextField1")
```

or shorter

```
oControlModel = oDialog.Model.TextField1
```

Dialog Properties

It is possible to make some modifications before a dialog is shown. An example is to set the dialog title that is shown in the title bar of a dialog window. This can be achieved by setting the Title property at the dialog model vthrough the `com.sun.star.beans.XPropertySet` interface:

```
oDialogModel = oDialog.getModel()
oDialogModel.setPropertyValue("Title", "My Title")
```

or shorter

```
oDialog.Model.Title = "My Title"
```

Another approach is to use the `setTitle` method of the `com.sun.star.awt.XDialog` interface:

```
oDialog.setTitle("My Title")
```

or

```
oDialog.Title = "My Title"
```

Another property is the BackgroundColor property that sets a different background color for the dialog.

Common Properties

All Toolkit control models have a set of identical properties referred as the *common properties*. These are the properties PositionX, PositionY, Width, Height, Name, TabIndex, Step and Tag.



Note that a Toolkit control model has those common properties only if it belongs to a dialog model. This has also some consequences for the creation of dialogs and controls at runtime. See *11.6 Basic and Dialogs - Creating Dialogs at Runtime.*

The PositionX, PositionY, Width and Height properties change the position and size of a dialog, and control at runtime. When designing a dialog in the dialog editor, these properties are set automatically.

The Name property is required, because all dialogs and controls are referenced by their name. In the dialog editor this name is created from the object name and a number, for example, TextField1.

The TabIndex property defines the order of focussing a control in a dialog when pressing the tabulator key. The index of the first element has the value 0. In the dialog editor the TabIndex property is set automatically when inserting a control. The order can also be changed through the property browser. Take care when setting this property at runtime.

The Tag property adds additional information to a control, such as a remark or number.

The Step property is described in detail in the next section.

Multi-Page Dialogs

A dialog may have several pages that can be traversed by the user step by step. This feature is used in the OpenOffice.org autopilots. The dialog property Step defines which page of the dialog is active. At runtime the next page of a dialog is displayed by increasing the step value by 1.

The Step property of a control defines the page of the dialog the control is visible. For example, if a control has a step value of 1, it is only visible on page 1 of the dialog. If the step value of the dialog is increased from 1 to 2, then all controls with a step value of 1 are faded out and all controls with a step value of 2 are visible.

A special role has the step value 0. For a control a step value of 0, the control is displayed on all dialog pages. If a dialog has a step value of 0, all controls of the dialog are displayed, independent of the step value of the single controls.

11.5.2 Dialog Controls

Command Button

The command button `com.sun.star.awt.UnoControlButton` allows the user to perform an action by clicking the button. Usually a button carries a label that is set through the Label property of the control model:

```
oDialogModel = oDialog.getModel()
oButtonModel = oDialogModel.getByName("CommandButton1")
oButtonModel.setPropertyValue("Label", "My Label")
```

or in short:

```
oDialog.Model.CommandButton1.Label = "My Label"
```

The label can also be set using the `setLabel` method of the `com.sun.star.awt.XButton` interface:

```
oButton = oDialog.getControl("CommandButton1")
oButton.setLabel("My Label")
```

During runtime, you may want to enable or disable a button. This is achieved by setting the Enabled property to `True` or `False`. The PushButtonType property defines the default action of a button where 0 is the Default, 1 is OK, 2 is Cancel, and 3 is Help. If a button has a PushButtonType value of 2, it behaves like a cancel button, that is, pressing the button closes the dialog. In this case, the method `execute()` of the dialog returns with a value of 0. An **OK** button of PushButtonType 1 returns 1 on `execute()`. The property DefaultButton specifies that the command button is the default button on the dialog, that is, pressing the ENTER key chooses the button even if another control has the focus. The Tabstop property defines if a control can be reached with the **TAB** key.

The command button has the feature, to display an image by setting the ImageURL property, which contains the path to the graphics file.

```
oButtonModel = oDialog.Model.CommandButton1
oButtonModel.ImageURL = "file:///D:/Office60/share/gallery/bullets/bluball.gif"
oButtonModel.ImageAlign = 2
```

All standard graphics formats are supported, such as *.gif*, *.jpg*, *.tif*, *.wmf* and *.bmp*. The property ImageAlign defines the alignment of the image inside the button where 0 is Left, 1 is Top, 2 is Right, and 3 is the Bottom. If the size of the image exceeds the size of the button, the image is not scaled automatically, but cut off. In this respect, the image control offers more functionality.

Image Control

If the user wants to display an image without the button functionality, the image control `com.sun.star.awt.UnoControlImageControl` is selected. The location of the graphic for the command button is set by the ImageURL property. Usually, the size of the image does not match the size of the control, therefore the image control automatically scales the image to the size of the control by setting the ScaleImage property to `True`.

```
oImageControlModel = oDialog.Model.ImageControl1
oImageControlModel.ImageURL = "file:///D:/Office60/share/gallery/photos/beach.jpg"
```

```
oImageControlModel.ScaleImage = True
```

Check Box

The check box control `com.sun.star.awt.UnoControlCheckBox` is used in groups to display multiple choices so that the user can select one or more choices. When a check box is selected it displays a check mark. Check boxes work independently of each other, thus different from option buttons. A user can select any number of check boxes at the same time.

The property State, where 0 is not checked, 1 is checked, 2 is don't know, accessesand changes the state of a checkbox. The tri-state mode of a check box is enabled by setting the TriState property to `True`. A tri-state check box provides the additional state "don't know", that is used to give the user the option of setting or unsetting an option.

```
oCheckBoxModel = oDialog.Model.CheckBox3
oCheckBoxModel.TriState = True
oCheckBoxModel.State = 2
```

The same result is achieved by using the `com.sun.star.awt.XCheckBox` interface:

```
oCheckBox = oDialog.getControl("CheckBox3")
oCheckBox.enableTriState( True )
oCheckBox.setState( 2 )
```

Option Button

An option button control `com.sun.star.awt.UnoControlRadioButton` is a simple switch with two states, that is selected by the user. Usually option buttons are used in groups to display several options, that the user may select. While option buttons and check boxes seem to be similar, selecting one option button deselects all the other option buttons in the same group.



Note, that option buttons thatbelong to the same group must have consecutive tab indices. Two groups of option buttons can be separated by any control with a tab index that is between the tab indices of the two groups.

Usually a group box, or horizontal and vertical lines are used, because those controls visually group the option buttons together, but in principal this can be any control. There is no functional relationship between an option button and a group box. Option buttons are grouped through consecutive tab indices only.

The state of an option button is accessed by the `State` property, where 0 is not checked and 1 is checked.

```
Function IsChecked( oOptionButtonModel As Object ) As Boolean

    Dim bChecked As Boolean

    If oOptionButtonModel.State = 1 Then
        bChecked = True
    Else
        bChecked = False
    End If

    IsChecked = bChecked

End Function
```

Label Field

A label field control `com.sun.star.awt.UnoControlFixedText` displays text that the user can no edit on the screen. For example, the label field is used to add descriptive labels to text fields, list boxes, and combo boxes. The actual text displayed in the label field is controlled by the `Label`

property. The Align property allows the user to set the alignment of the text in the control to the left (0), center (1) or right (2). By default, the label field displays the text from the Label property in a single line. If the text exceeds the width of the control, the text is truncated. This behavior is changed by setting the MultiLine property to True, so that the text is displayed on more than one line, if necessary. By default, the label field control is drawn without any border. However, the label field appears with a border if the Border property is set, where 0 is no border, 1 is a 3D border, and 2 is a simple border. The font attributes of the text in the label field are specified by the FontDescriptor property. It is recommended to set this property with the property browser in the dialog editor.

Label fields are used to define shortcut keys for controls without labels. A shortcut key can be defined for any control with a label by adding a tilde (~) before the character that will be used as a shortcut. When the user presses the character key simultaneously with the ALT key, the control automatically gets the focus. To assign a shortcut key to a control without a label, for example, a text field, the label field is used. The tilde prefixes the corresponding character in the Label property of the label field. As the label field cannot receive focus, the focus automatically moves to the next control in the tab order. Therefore, it is important that the label field and the text field have consecutive tab indices.

```
oLabelModel = oDialog.Model.Label1
oLabelModel.Label = "Enter ~Text"
```

Text Field

The text field control com.sun.star.awt.UnoControlEdit is used to get input from the user at runtime. In general, the text field is used for editable text, but it can also be made read-only by setting the ReadOnly property to True. The actual text displayed in a text field is controlled by the Text property. The maximum number of characters that can be entered by the user is specified with the MaxTextLen property. A value of 0 means that there is no limitation. By default, a text field displays a single line of text. This behavior is changed by setting the property MultiLine to True. The properties HScroll and VScroll displays a horizontal and vertical scroll bar.

When a text field receives the focus by pressing the **TAB** key the displayed text is selected and highlighted by default. The default cursor position within the text field is to the right of the existing text. If the user starts typing while a block of text is selected, the selected text is replaced. In some cases, the user may change the default selection behavior and set the selection manually. This is done using the com.sun.star.awt.XTextComponent interface:

```
Dim sText As String
Dim oSelection As New com.sun.star.awt.Selection

REM get control
oTextField = oDialog.getControl("TextField1")

REM set displayed text
sText = "Displayed Text"
oTextField.setText( sText )

REM set selection
oSelection.Min = 0
oSelection.Max = Len( sText )
oTextField.setSelection( oSelection )
```

The text field control is also used for entering passwords. The property EchoChar specifies the character that is displayed in the text field while the user enters the password. In this context, the MaxTextLen property is used to limit the number of characters that are typed in:

```
oTextFieldModel = oDialog.Model.TextField1
oTextFieldModel.EchoChar = Asc("*")
oTextFieldModel.MaxTextLen = 8
```

A user can enter any kind of data into a text field, such as numerical values and dates. These values are always stored as a string in the Text property, thus leadingto problems when evalu-

ating the user input. Therefore, consider using a date field, time field, numeric field, currency field or formatted field instead.

List Box

The list box control `com.sun.star.awt.UnoControlListBox` displays a list of items that the user can select one or more of. If the number of items exceeds what can be displayed in the list box, scroll bars automatically appear on the control. If the `Dropdown` property is set to `True`, the list of items is displayed in a drop-down box. In this case, the maximum number of line counts in the drop-down box are specified with the `LineCount` property. The actual list of items is controlled by the `StringItemList` property. All selected items are controlled by the `SelectedItems` property. If the `MultiSelection` property is set to `True`, more than one entry can be selected.

It may be easier to use the `com.sun.star.awt.XListBox` interface when working with list boxes, because an item can be added to a list at a specific position with the `addItem` method. For example, an item is added at the end of the list by:

```
Dim nCount As Integer

oListBox = oDialog.getControl("ListBox1")
nCount = oListBox.getItemCount()
oListBox.addItem( "New Item", nCount )
```

Multiple items are added with the help of the `addItems` method. The `removeItems` method is used to remove items from a list. For example, the first entry in a list is removed by:

```
Dim nPos As Integer, nCount As Integer

nPos = 0
nCount = 1
oListBox.removeItems( nPos, nCount )
```

A list box item can be preselected with the `selectItemPos`, `selectItemsPos` and `selectItem` methods. For example, the first entry in a list box can be selected by:

```
oListBox.selectItemPos( 0, True )
```

The currently selected item is obtained with the `getSelectedItem` method:

```
Dim sSelectedItem As String
sSelectedItem = oListBox.getSelectedItem()
```

Combo Box

The combo box control `com.sun.star.awt.UnoControlComboBox` presents a list of choices to the user. Additionally, it contains a text field allowing the user to input a selection that is not on the list. A combo box is used when there is only a list of suggested choices, whereas a list box is used when the user input is limited only to the list.

The features and properties of a combo box and a list box are similar. Also in a combo box the list of items can be displayed in a drop-down box by setting the Dropdown property to `True`. The actual list of items is accessible through the `StringItemList` property. The text displayed in the text field of the combo box is controlled by the `Text` property. For example, if a user selects an item from the list, the selected item is displayed in the text field and is obtained from the `Text` property:

```
Function GetSelectedItem( oComboBoxModel As Object ) As String
    GetSelectedItem = oComboBoxModel.Text
End Function
```

When a user types text into the text field of the combo box, the automatic word completion is a useful feature and is enabled by setting the `Autocomplete` property to `True`. It is recommended to use the `com.sun.star.awt.XComboBox` interface when accessing the items of a combo box:

```
Dim nCount As Integer
Dim sItems As Variant

REM get control
oComboBox = oDialog.getControl("ComboBox1")

REM first remove all old items from the list
nCount = oComboBox.getItemCount()
oComboBox.removeItems( 0, nCount )

REM add new items to the list
sItems = Array( "Item1", "Item2", "Item3", "Item4", "Item5" )
oComboBox.addItems( sItems, 0 )
```

Horizontal/Vertical Scroll Bar

If the visible area in a dialog is smaller than the displayable content, the scroll bar control `com.sun.star.awt.UnoControlScrollBar` provides navigation through the content by scrolling horizontally or vertically. In addition, the scroll bar control is used to provide scrolling to controls that do not have a built-in scroll bar.

The orientation of a scroll bar is specified by the Orientation property and can be horizontal or vertical. A scroll bar has a thumb (scroll box) that the user can drag with the mouse to any position along the scroll bar. The position of the thumb is controlled by the `ScrollValue` property. For a horizontal scroll bar, the left-most position corresponds to the minimum scroll value of 0 and the right-most position to the maximum scroll value defined by the `ScrollValueMax` property. A scroll bar also has arrows at its end that when clicked or held, incrementally moves the thumb along the scroll bar to increase or decrease the scroll value. The change of the scroll value per mouse click on an arrow is specified by the `LineIncrement` property. When clicking in a scroll bar in the region between the thumb and the arrows, the scroll value increases or decreases by the value set for the `BlockIncrement` property. The thumb position represents the portion of the displayable content that is currently visible in a dialog. The visible size of the thumb is set by the `VisibleSize` property and represents the percentage of the currently visible content and the total displayable content.

```
oScrollBarModel = oDialog.Model.ScrollBar1
oScrollBarModel.ScrollValueMax = 100
oScrollBarModel.BlockIncrement = 20
oScrollBarModel.LineIncrement = 5
oScrollBarModel.VisibleSize = 20
```

The scroll bar control uses the adjustment event `com.sun.star.awt.AdjustmentEvent` to monitor the movement of the thumb along the scroll bar. In an event handler for adjustment events the developer may change the position of the visible content on the dialog as a function of the `ScrollValue` property. In the following example, the size of a label field exceeds the size of the dialog. Each time the user clicks on the scrollbar, the macro `AdjustmentHandler()` is called and the position of the label field in the dialog is changed according to the scroll value. (BasicAndDialogs/ToolkitControls/ScrollBar.xba)

```
Sub AdjustmentHandler()

    Dim oLabelModel As Object
    Dim oScrollBarModel As Object
    Dim ScrollValue As Long, ScrollValueMax As Long
    Dim VisibleSize As Long
    Dim Factor As Double

    Static bInit As Boolean
    Static PositionX0 As Long
    Static Offset As Long

    REM get the model of the label control
    oLabelModel = oDialog.Model.Label1

    REM on initialization remember the position of the label control and calculate offset
    If bInit = False Then
        bInit = True
        PositionX0 = oLabelModel.PositionX
        OffSet = PositionX0 + oLabelModel.Width - (oDialog.Model.Width - Border)
```

```
    End If

    REM get the model of the scroll bar control
    oScrollBarModel = oDialog.Model.ScrollBar1

    REM get the actual scroll value
    ScrollValue = oScrollBarModel.ScrollValue

    REM calculate and set new position of the label control
    ScrollValueMax = oScrollBarModel.ScrollValueMax
    VisibleSize = oScrollBarModel.VisibleSize
    Factor = Offset / (ScrollValueMax - VisibleSize)
    oLabelModel.PositionX = PositionX0 - Factor * ScrollValue

End Sub
```

Group Box

The group box control `com.sun.star.awt.UnoControlGroupBox` creates a frame to visually group other controls together, such as option buttons and check boxes. Note that the group box control does not provide any container functionality for other controls, it only has visual functionality. For more details, see *11.5.2 Basic and Dialogs - Programming Dialogs and Dialog Controls - Dialog Controls - Option Button.*

The group box contains a label embedded within the border and is set by the Label property. In most cases, the group box control is only used passively.

Progress Bar

The progress bar control `com.sun.star.awt.UnoControlProgressBar` displays a growing or shrinking bar to give the user feedback during an operation, for example, the completion of a lengthy task. The minimum and the maximum progress value of the control is set by the `ProgressValueMin` and the `ProgressValueMax` properties. The progress value is controlled by the `ProgressValue` property. By default, the progress bar is blue, but the fill color can be changed by setting the `FillColor` property. The functionality of a progress bar is demonstrated in the following example: (BasicAndDialogs/ToolkitControls/ProgressBar.xba)

```
Sub ProgressBarDemo()

    Dim oProgressBar As Object, oProgressBarModel As Object
    Dim oCancelButtonModel As Object
    Dim oStartButtonModel As Object
    Dim ProgressValue As Long

    REM progress bar settings
    Const ProgressValueMin = 0
    Const ProgressValueMax = 40
    Const ProgressStep = 4

    REM set minimum and maximum progress value
    oProgressBarModel = oDialog.Model.ProgressBar1
    oProgressBarModel.ProgressValueMin = ProgressValueMin
    oProgressBarModel.ProgressValueMax = ProgressValueMax

    REM disable cancel and start button
    oCancelButtonModel = oDialog.Model.CommandButton1
    oCancelButtonModel.Enabled = False
    oStartButtonModel = oDialog.Model.CommandButton2
    oStartButtonModel.Enabled = False

    REM show progress bar
    oProgressBar = oDialog.getControl("ProgressBar1")
    oProgressBar.setVisible( True )

    REM increase progress value every second
    For ProgressValue = ProgressValueMin To ProgressValueMax Step ProgressStep
        oProgressBarModel.ProgressValue = ProgressValue
        Wait 1000
    Next ProgressValue

    REM hide progress bar
    oProgressBar.setVisible( False )
```

```
    REM enable cancel and start button
    oCancelButtonModel.Enabled = True
    oStartButtonModel.Enabled = True

End Sub
```

Horizontal/Vertical Line

The line control `com.sun.star.awt.UnoControlFixedLine` creates simple lines in a dialog. In most cases, the line control is used to visually subdivide a dialog. The line control can have horizontal or vertical orientation that is specified by the `Orientation` property. The label of a line control is set by the `Label` property. Note that the label is only displayed if the control has a horizontal orientation.

Date Field

The date field control `com.sun.star.awt.UnoControlDateField` extends the text field control and is used for displaying and entering dates. The date displayed in the date field is controlled by the Date property. The date value is of type `Long` and must be specified in the format YYYYMMDD, for example, the date September 30th, 2002 is set in the following format:

```
oDateFieldModel = oDialog.Model.DateField1
oDateFieldModel.Date = 20020930
```

The current date is set by using the `Date` and `CDateToIso` runtime functions:

```
oDateFieldModel.Date = CDateToIso( Date() )
```

The minimum and the maximum date that the user can enter is defined by the `DateMin` and the `DateMax` property. The format of the displayed date is specified by the `DateFormat` and the `DateShowCentury` property, but the usage of `DateShowCentury` is deprecated. Some formats are dependent on the system settings. If the `StrictFormat` property is set to `True`, the date entered by the user is checked during input. The `Dropdown` property enables a calendar that the user can drop down to select a date.

Dropdown is currently not working.

Time Field

The time field control `com.sun.star.awt.UnoControlDateField` displays and enters time values. The time value are set and retrieved by the `Time` property. The time value is of type `Long` and is specified in the format HHMMSShh, where HH are hours, MM are minutes, SS are seconds and hh are hundredth seconds. For example, the time 15:18:23 is set by:

```
oTimeFieldModel = oDialog.Model.TimeField1
oTimeFieldModel.Time = 15182300
```

The minimum and maximum time value that can be entered is given by the `TimeMin` and `TimeMax` property. The format of the displayed time is specified by the `TimeFormat` property.

The time value is checked during input by setting the `StrictFormat` property to `True`.

Short time format is currently not working.

Numeric Field

It is recommended to use the numeric field control `com.sun.star.awt.UnoControlNumericField` if the user input is limited to numeric values. The numeric value is controlled by the Value property, which is of type `Double`. A minimum and maximum value for user input is defined by the `ValueMin` and the `ValueMax` property. The decimal accuracy of the numeric value is specified by the `DecimalAccuracy` property, for example, a value of 6 corresponds to 6 decimal places. If the `ShowThousandsSeparator` property is set to `True`, a thousands separator is displayed. The numeric field also has a built-in spin button, enabled by the `Spin` property. The spin button is used to increment and decrement the displayed numeric value by clicking with the mouse, whereas the step is set by the `ValueStep` property.

```
oNumericFieldModel = oDialog.Model.NumericField1
oNumericFieldModel.Value = 25.40
oNumericFieldModel.DecimalAccuracy = 2
```

Currency Field

The currency field control `com.sun.star.awt.UnoControlCurrencyField` is used for entering and displaying currency values. In addition to the currency value, a currency symbol is displayed, that is set by the `CurrencySymbol` property. If the `PrependCurrencySymbol` property is set to `True`, the currency symbol is displayed in front of the currency value.

```
oCurrencyFieldModel = oDialog.Model.CurrencyField1
oCurrencyFieldModel.Value = 500.00
oCurrencyFieldModel.CurrencySymbol = "€"
oCurrencyFieldModel.PrependCurrencySymbol = True
```

Formatted Field

The formatted field control `com.sun.star.awt.UnoControlFormattedField` specifies a format that is used for formatting the entered and displayed data. A number formats supplier must be set in the `FormatsSupplier` property and a format key for the used format must be specified in the `FormatKey` property. It is recommended to use the property browser in the dialog editor for setting these properties. Supported number formats are number, percent, currency, date, time, scientific, fraction and boolean values. Therefore, the formatted field can be used instead of a date field, time field, numeric field or currency field. The NumberFormatsSupplier is described in *6 Office Development*.

Pattern Field

The pattern field control `com.sun.star.awt.UnoControlPatternField` displays and enters a string according to a specified pattern. The entries that the user enters in the pattern field are defined in the EditMask property as a special character code. The length of the edit mask determines the number of the possible input positions. If a character is entered that does not correspond to the edit mask, the input is rejected. For example, in the edit mask "NNLNNLLLLL" the character L has the meaning of a text constant and the character N means that only the digits 0 to 9 can be entered. A complete list of valid characters can be found in the OpenOffice.org online help. The LiteralMask property contains the initial values that are displayed in the pattern field. The length of the literal mask should always correspond to the length of the edit mask. An example of a literal mask which fits to the above mentioned edit mask would be "__.__.2002". In this case, the user enters only 4 digits when entering a date.

```
oPatternFieldModel = oDialog.Model.PatternField1
oPatternFieldModel.EditMask    = "NNLNNLLLLL"
oPatternFieldModel.LiteralMask = "__.__.2002"
```

File Control

The file control `com.sun.star.awt.UnoControlFileControl` has all the properties of a text field control, with the additional feature of a built-in command button. When the button is clicked, the file dialog shows up. The directory that the file dialog initially displays is set by the Text property.

The directory must be given as a system path, file URLs do not work at the moment. In Basic you can use the runtime function `ConvertToURL()` to convert system paths to URLs.

```
oFileControl = oDialog.Model.FileControl1
oFileControl.Text = "D:\Programme\Office60"
```

Filters for the file dialog can not be set or appended for the file control. An alternative way is to use a text field and a command button instead of a file control and assign a macro to the button which instantiates the file dialog `com.sun.star.ui.dialogs.FilePicker` at runtime. An example is provided below. (BasicAndDialogs/ToolkitControls/FileDialog.xba)

```
Sub OpenFileDialog()

    Dim oFilePicker As Object, oSimpleFileAccess As Object
    Dim oSettings As Object, oPathSettings As Object
    Dim oTextField As Object, oTextFieldModel As Object
    Dim sFileURL As String
    Dim sFiles As Variant

    REM file dialog
    oFilePicker = CreateUnoService( "com.sun.star.ui.dialogs.FilePicker" )

    REM set filter
    oFilePicker.AppendFilter( "All files (*.*)", "*.*" )
    oFilePicker.AppendFilter( "StarOffice 6.0 Text Text Document", "*.sxw" )
    oFilePicker.AppendFilter( "StarOffice 6.0 Spreadsheet", "*.sxc" )
    oFilePicker.SetCurrentFilter( "All files (*.*)" )

    REM if no file URL is set, get path settings from configuration
    oTextFieldModel = oDialog.Model.TextField1
    sFileURL = ConvertToURL( oTextFieldModel.Text )
    If sFileURL = "" Then
        oSettings = CreateUnoService( "com.sun.star.frame.Settings" )
        oPathSettings = oSettings.getByName( "PathSettings" )
        sFileURL = oPathSettings.getPropertyValue( "Work" )
    End If

    REM set display directory
    oSimpleFileAccess = CreateUnoService( "com.sun.star.ucb.SimpleFileAccess" )
    If oSimpleFileAccess.exists( sFileURL ) And oSimpleFileAccess.isFolder( sFileURL ) Then
        oFilePicker.setDisplayDirectory( sFileURL )
    End If

    REM execute file dialog
    If oFilePicker.execute() Then
        sFiles = oFilePicker.getFiles()
        sFileURL = sFiles(0)
        If oSimpleFileAccess.exists( sFileURL ) Then
            REM set file path in text field
            oTextField = oDialog.GetControl("TextField1")
            oTextField.SetText( ConvertFromURL( sFileURL ) )
        End If
    End If

End Sub
```

11.6 Creating Dialogs at Runtime

When using OpenOffice.org Basic, the dialog editor is a tool for designing dialogs. Refer to *11.2 Basic and Dialogs - OpenOffice.org Basic IDE* for additional information. When using Java, a different approach is used, because Java is not supported as scripting language. Dialogs are created at runtime in a similar method as Java Swing components are created. Also, the event listeners are registered at runtime at the appropriate controls.

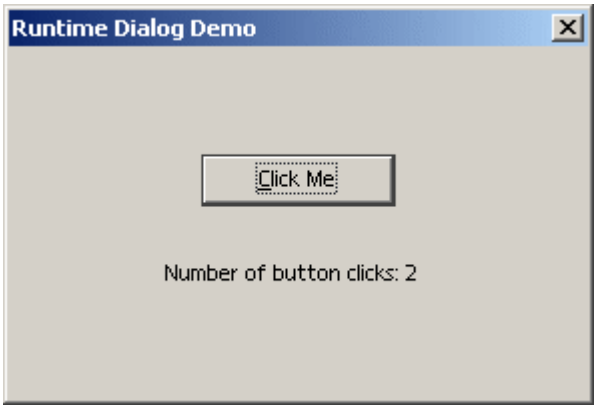


Illustration 169

In the example described in this section, a simple modal dialog is created at runtime containing a command button and label field. Each time the user clicks on the button, the label field is updated and the total number of button clicks is displayed.

The dialog is implemented as a UNO component in Java that is instantiated with the service name `com.sun.star.examples.SampleDialog`. For details about writing a Java component and the implementation of the UNO core interfaces, refer to *4.5.6 Writing UNO Components - Simple Component in Java - Storing the Service Manager for Further Use*. The method that creates and executes the dialog is shown below.

```
/** method for creating a dialog at runtime
 */
private void createDialog() throws com.sun.star.uno.Exception {

    // get the service manager from the component context
    XMultiComponentFactory xMultiComponentFactory = _xComponentContext.getServiceManager();

    // create the dialog model and set the properties
    Object dialogModel = xMultiComponentFactory.createInstanceWithContext(
        "com.sun.star.awt.UnoControlDialogModel", _xComponentContext);
    XPropertySet xPSetDialog = (XPropertySet)UnoRuntime.queryInterface(
        XPropertySet.class, dialogModel);
    xPSetDialog.setPropertyValue("PositionX", new Integer(100));
    xPSetDialog.setPropertyValue("PositionY", new Integer(100));
    xPSetDialog.setPropertyValue("Width", new Integer(150));
    xPSetDialog.setPropertyValue("Height", new Integer(100));
    xPSetDialog.setPropertyValue("Title", new String("Runtime Dialog Demo"));

    // get the service manager from the dialog model
    XMultiServiceFactory xMultiServiceFactory = (XMultiServiceFactory)UnoRuntime.queryInterface(
        XMultiServiceFactory.class, dialogModel);

    // create the button model and set the properties
    Object buttonModel = xMultiServiceFactory.createInstance(
        "com.sun.star.awt.UnoControlButtonModel" );
    XPropertySet xPSetButton = (XPropertySet)UnoRuntime.queryInterface(
        XPropertySet.class, buttonModel);
    xPSetButton.setPropertyValue("PositionX", new Integer(50));
    xPSetButton.setPropertyValue("PositionY", new Integer(30));
    xPSetButton.setPropertyValue("Width", new Integer(50));
    xPSetButton.setPropertyValue("Height", new Integer(14));
    xPSetButton.setPropertyValue("Name", _buttonName);
    xPSetButton.setPropertyValue("TabIndex", new Short((short)0));
    xPSetButton.setPropertyValue("Label", new String("Click Me"));

    // create the label model and set the properties
    Object labelModel = xMultiServiceFactory.createInstance(
        "com.sun.star.awt.UnoControlFixedTextModel" );
    XPropertySet xPSetLabel = ( XPropertySet )UnoRuntime.queryInterface(
        XPropertySet.class, labelModel );
    xPSetLabel.setPropertyValue("PositionX", new Integer(40));
    xPSetLabel.setPropertyValue("PositionY", new Integer(60));
    xPSetLabel.setPropertyValue("Width", new Integer(100));
    xPSetLabel.setPropertyValue("Height", new Integer(14));
    xPSetLabel.setPropertyValue("Name", _labelName);
    xPSetLabel.setPropertyValue("TabIndex", new Short((short)1));
    xPSetLabel.setPropertyValue("Label", _labelPrefix);
```

```
    // insert the control models into the dialog model
    XNameContainer xNameCont = (XNameContainer)UnoRuntime.queryInterface(
        XNameContainer.class, dialogModel);
    xNameCont.insertByName(_buttonName, buttonModel);
    xNameCont.insertByName(_labelName, labelModel);

    // create the dialog control and set the model
    Object dialog = xMultiComponentFactory.createInstanceWithContext(
        "com.sun.star.awt.UnoControlDialog", _xComponentContext);
    XControl xControl = (XControl)UnoRuntime.queryInterface(
        XControl.class, dialog );
    XControlModel xControlModel = (XControlModel)UnoRuntime.queryInterface(
        XControlModel.class, dialogModel);
    xControl.setModel(xControlModel);

    // add an action listener to the button control
    XControlContainer xControlCont = (XControlContainer)UnoRuntime.queryInterface(
        XControlContainer.class, dialog);
    Object objectButton = xControlCont.getControl("Button1");
    XButton xButton = (XButton)UnoRuntime.queryInterface(XButton.class, objectButton);
    xButton.addActionListener(new ActionListenerImpl(xControlCont));

    // create a peer
    Object toolkit = xMultiComponentFactory.createInstanceWithContext(
        "com.sun.star.awt.ExtToolkit", _xComponentContext);
    XToolkit xToolkit = (XToolkit)UnoRuntime.queryInterface(XToolkit.class, toolkit);
    XWindow xWindow = (XWindow)UnoRuntime.queryInterface(XWindow.class, xControl);
    xWindow.setVisible(false);
    xControl.createPeer(xToolkit, null);

    // execute the dialog
    XDialog xDialog = (XDialog)UnoRuntime.queryInterface(XDialog.class, dialog);
    xDialog.execute();

    // dispose the dialog
    XComponent xComponent = (XComponent)UnoRuntime.queryInterface(XComponent.class, dialog);
    xComponent.dispose();
}
```

First, a dialog model is created by prompting the ServiceManager for the `com.sun.star.awt.UnoControlDialogModel` service. Then, the position, size and title of the dialog are set using the `com.sun.star.beans.XPropertySet` interface. In performance critical applications, the use of the `com.sun.star.beans.XMultiPropertySet` interface is recommended. At this point, the dialog model describes an empty dialog, which does not contain any control models.

All control models in a dialog container have the common properties “PositionX”, “PositionY”, “Width”, “Height”, “Name”, “TabIndex”, “Step” and “Tag”. These properties are optional and only added if the control model is created by a special object factory, namely the dialog model. Therefore, a dialog model also supports the `com.sun.star.lang.XMultiServiceFactory` interface. If the control model is created by the ServiceManager, these common properties are missing.



Note that control models have the common properties “PositionX”, “PositionY”, “Width”, “Height”, “Name”, “TabIndex”, “Step” and “Tag” only if they were created by the dialog model that they belong to.

After the control models for the command button and label field are created, their position, size, name, tab index and label are set. Then, the control models are inserted into the dialog model using the `com.sun.star.container.XNameContainer` interface. The model of the dialog has been fully described.

To display the dialog on the screen, a dialog control `com.sun.star.awt.UnoControlDialog` is created and the corresponding model is set. An action listener is added to the button control, because the label field is updated whenever the user clicks on the command button. The listener is explained below. Before the dialog is shown, a window or a *peer* is created on the screen. Finally, the dialog is displayed on the screen using the `execute` method of the `com.sun.star.awt.XDialog` interface.

The implementation of the action listener is shown in the following example.

```
/** action listener
```

```
 */
public class ActionListenerImpl implements com.sun.star.awt.XActionListener {
    private int _nCounts = 0;
    private XControlContainer _xControlCont;

    public ActionListenerImpl(XControlContainer xControlCont) {
        _xControlCont = xControlCont;
    }

    // XEventListener
    public void disposing(EventObject eventObject) {
        _xControlCont = null;
    }

    // XActionListener
    public void actionPerformed(ActionEvent actionEvent) {
        // increase click counter
        _nCounts++;

        // set label text
        Object label = _xControlCont.getControl("Label1");
        XFixedText xLabel = (XFixedText)UnoRuntime.queryInterface(XFixedText.class, label);
        xLabel.setText(_labelPrefix + _nCounts);
    }
}
```

The action listener is fired each time the user clicks on the command button. In the `actionPerformed` method of the `com.sun.star.awt.XActionListener` interface, an internal counter for the number of button clicks is increased. Then, this number is updated in the label field. In addition, the `disposing` method of the parent interface `com.sun.star.lang.XEventListener` is implemented.

Our sample component executes the dialog from within the office by implementing the `trigger` method of the `com.sun.star.task.XJobExecutor` interface:

```
public void trigger(String sEvent) {
    if (sEvent.compareTo("execute") == 0) {
        try {
            createDialog();
        }
        catch (Exception e) {
            throw new com.sun.star.lang.WrappedTargetRuntimeException(e.getMessage(), this, e);
        }
    }
}
```

A simple OpenOffice.org Basic macro that instantiates the service of our sample component and executes the dialog is shown below.

```
Sub Main
    Dim oJobExecutor
    oJobExecutor = CreateUnoService("com.sun.star.examples.SampleDialog")
    oJobExecutor.trigger("execute")
End Sub
```

In future versions of OpenOffice.org, a method for executing dialogs created at runtime will be provided.

11.7 Library File Structure

This section describes how libraries are stored. Generally all data is stored in XML format. Four different XML document types that arespecified in the DTD files installed in *<OfficePath>/share/dtd/officedocument* are used:

- A library container is described by a library container index file following the specification given in *libraries.dtd*. In this file, each library in the library container is described by its name, a flag if the library is a link, the StorageURL (describing where the library is stored) and, only in case of a link, the link read-only status.

- A library is described by a library index file following the specification given in *library.dtd*. This file contains the library name, a flag for the read-only status, a flag if the library is password protected (see below) and the name of each library element.
- A Basic source code module is described in a file following the specification given in *module.dtd*. This file contains the module name, the language (at the moment only OpenOffice.org Basic is supported) and the source code.
- A dialog is described in a file following the specification given in *dialog.dtd*. The file contains all data to describe a dialog. As this format is extensive, it is not possible to describe it in this document.

Additionally, a binary format is used to store compiled Basic code for password protected Basic libraries. This is described in more detail in *11.7 Basic and Dialogs - Library File Structure*.



In a password protected Basic library, the password is used to scramble the source code using the Blowfish algorithm. The password itself is not stored, so when the password for a Basic library is lost, the corresponding Basic source code is lost also. There is *no* retrieval method if this happens.

Besides the XML format of the library description files, it is necessary to understand the structure in which these files are stored. This is different for application and document libraries. Application libraries are stored directly in the system file system and document libraries are stored inside the document's package file. For information abuot package files, see *6.2.10 Office Development - Common Application Features - Package File Formats*. The following sections describe the structure and combination of library container and library structures.

11.7.1 Application Library Container

In an OpenOffice.org installation the application library containers for Basic and dialogs are located in the directory *<OfficePath>/user/basic*. The library container index files are named *script.xlc* for the Basic and *dialog.xlc* for the Dialog library container. The "lc" in *.xlc* stands for library container.

The same directory contains the libraries created by the user. Initially only the library Standard exists for Basic and dialogs using the same directory. The structure of the library inside the directory is explained in the next section.

The *user/basic* directory is not the only place in the OpenOffice.org installation where libraries are stored. Most of the autopilots integrated in OpenOffice.org are realized in Basic, and the corresponding Basic and dialog libraries are installed in the directory *<OfficePath>/share/basic*. These libraries are listed in the library container index file as read-only links.

It is necessary to distinguish between libraries created by the user and the autopilot libraries. The autopilot libraries are installed in a directory that is shared between different users. In a network installation, the *share* directory is located somewhere on a server, so that the autopilot libraries cannot be owned directly by the user-specific library containers.

In the file system, a library is represented by a directory. The directory's name is the same as the library name. The directory contains all files that are necessary for the library.

Basic libraries can be protected with a password, so that the source code cannot be read by unauthorized persons. Dialog libraries cannot be protected with a password. This can be handled using the **Tools - Macro - Organizer...** dialog that is explained in *11.2.1 Basic and Dialogs - OpenOffice.org Basic IDE - Managing Basic and Dialog Libraries*. The password protection of a Basic library also affects the file format.

Libraries without Password Protection

Every library element is represented by an XML file named like the element in the directory representing the library. For Basic modules these files, following the specification in *module.dtd,* have the extension *.xba*. For dialogs these files, following the specification in *dialog.dtd,* have the extension *.xdl*. Additionally, the directory contains a library index file (*library.dtd*). These index files are named *script.xlb* for Basic and *dialog.xlb* for dialog libraries.

In the following example, an Application Basic library Standard containing two modules Module1 and Module2 is represented by the following directory:

```
<DIR> Standard
|
|--script.xlb
|--Module1.xba
|--Module2.xba
```

An application dialog library Standard containing two dialogs SmallDialog and BigDialog is represented by the following directory:

```
<DIR> Standard
|
|--dialog.xlb
|--SmallDialog.xba
|--BigDialog.xba
```

It is also possible that the same directory represents a Basic and a Dialog library. This is the standard case in the OpenOffice.org, See the chapter Library organization in OpenOffice.org. When the two example libraries above are stored in the same directory, the files from both libraries are together in the same directory:

```
<DIR> Standard
|
|--dialog.xlb
|--script.xlb
|--Module1.xba
|--Module2.xba
|--SmallDialog.xba
|--BigDialog.xba
```

The two libraries do not affect each other, because all file names are different. This is also the case if a Basic module and a dialog are named equally, due the different file extensions..

Libraries with Password Protection

Only Basic libraries can be password protected. The password protection of a Basic library affects the file format, because binary data has to be stored. In plain XML format, the source code would be readable in the file even if it was not displayed in the Basic IDE. Also, the compiled Basic code has to be stored for each module together with the encrypted sources. This is necessary because, Basic could not access the source code and compile it as long as the password is unknown in contrast to libraries without password protection. Without storing the compiled code, Basic could only execute password-protected libraries once the user supplied the correct password. The whole purpose of the password feature is to distribute programs without giving away the password and source code, therefore this would not be feasible.

The followig example shows a password-protected application Basic library Library1, containing three modules Module1, Module1 and Module3, is represented by the following directory:

```
<DIR> Library1
|
|--script.xlb
|--Module1.pba
|--Module2.pba
|--Module3.pba
```

The file *script.xlb* does not differ from the case without a password, except for the fact that the password protected status of the library is reflected by the corresponding flag.

Each module is represented by a *.pba* file. Like OpenOffice.org documents, these files are package files ("pba" stands for *p*ackage *ba*sic) and contain a sub structure that can be viewed with any zip tool. For detailed information about package files, see *6.2.10 Office Development - Common Application Features - Package File Formats*).

A module package file has the following content:

```
<PACKAGE> Module1.pba
|
|--<DIR> Meta-Inf            ' Content is not displayed here
|--code.bin
|--source.xml
```

The *Meta-Inf* directory is part of every package file and will not be explained in this document. The file *code.bin* contains the compiled Basic code and the file *source.xml* contains the Basic source code encrypted with the password.

11.7.2 Document Library Container

While application libraries are stored directly in the file system, document libraries are stored inside the document's package file. For more informatin about package files, see *6.2.10 Office Development - Common Application Features - Package File Formats*. In documents, the Basic library container and dialog library container are stored separately:

- The root of the Basic library container hierarchy is a folder inside the package file named *Basic*. This folder is not created when the Basic library container contains an empty Standard library in the case of a new document.
- The root of the dialog library container hierarchy is a folder inside the package file named *Dialogs*. This folder is not created when the dialog library container contains an empty Standard library in the case of a new document.

The libraries are stored as sub folders in these library container folders. The structure inside the libraries is basically the same as in an application. One difference relates to the stream - "files" inside the package or package folders – names. In documents, all XML stream or file names have the extension *.xml*. Special extensions like *.xba*, *.xdl* are not used. Instead of different extensions, the names are extended for the library and library container index files. In documents they are named *script-lc.xml* (Basic library container index file), *script-lb.xml* (Basic library index file), *dialog-lc.xml* (dialog library container index file) and *dialog-lb.xml* (dialog library index file).

In example 1, the package structure for a document with one Basic Standard library containing three modules:

```
<Package> ExampleDocument1
|
|--<DIR> Basic
|  |
|  |--<DIR> Standard         ' Folder: Contains library "Standard"
|  |  |
|  |  |--Module1.xml         ' Stream: Basic module file
|  |  |--Module2.xml         ' Stream: Basic module file
|  |  |--Module3.xml         ' Stream: Basic module file
|  |  |--script-lb.xml       ' Stream: Basic library index file
|  |
|  |--script-lc.xml          ' Stream: Basic library container index file
|
|  ' From here the folders and streams have nothing to do with libraries
|--<DIR> Meta-Inf
|--content.xml
|--settings.xml
|--styles.xml
```

In example 2, package structure for a document with two Basic and one dialog libraries:

```
<Package> ExampleDocument2
|
```

```
|--<DIR> Basic
|  |
|  |--<DIR> Standard          ' Folder: Contains library "Standard"
|  |  |
|  |  |--Module1.xml          ' Stream: Basic module file
|  |  |--Module2.xml          ' Stream: Basic module file
|  |  |--script-lb.xml        ' Stream: Basic library index file
|  |
|  |--<DIR> Library1          ' Folder: Contains library "Library1"
|  |  |
|  |  |--Module1.xml          ' Stream: Basic module file
|  |  |--script-lb.xml        ' Stream: Basic library index file
|  |
|  |--script-lc.xml           ' Stream: Basic library container index file
|
|--<DIR> Dialogs
|  |
|  |--<DIR> Standard          ' Folder: Contains library "Standard"
|  |  |
|  |  |--Dialog1.xml          ' Stream: Dialog file
|  |  |--dialog-lb.xml        ' Stream: Dialog library index file
|  |
|  |--<DIR> Library1          ' Folder: Contains library "Library1"
|  |  |
|  |  |--Dialog1.xml          ' Stream: Dialog file
|  |  |--Dialog2.xml          ' Stream: Dialog file
|  |  |--dialog-lb.xml        ' Stream: Dialog library index file
|  |
|  |--dialog-lc.xml           ' Stream: Dialog library container index file
|
|  ' From here the folders and streams have nothing to do with libraries
|--<DIR> Meta-Inf
|--content.xml
|--settings.xml
|--styles.xml
```

If a document Basic library is password protected, the file structure does not differ as much from an unprotected library as in the Application Basic case. The differences are:

- The module files of a password-protected Basic library have the same name as without the password protection, but they are scrambled with the password.
- There is an additional binary file named like the library with the extension *.bin* for each module. Similar to the file *code.bin* in the Application Basic *.pba* files, this file contains the compiled Basic code that executes the module without access to the source code.

The following example shows the package structure for a document with two Basic and one dialog libraries where only the Basic library Library1 contains any of the modules:

```
<Package> ExampleDocument3
|
|--<DIR> Basic
|  |
|  |--<DIR> Standard          ' Folder: Contains library "Standard"
|  |  |
|  |  |--script-lb.xml        ' Stream: Basic library index file
|  |
|  |--<DIR> Library1          ' Folder: Contains library "Library1"
|  |  |
|  |  |--Module1.xml          ' Stream: Scrambled Basic module source file
|  |  |--Module1.bin          ' Stream: Basic module compiled code file
|  |  |--Module2.xml          ' Stream: Scrambled Basic module source file
|  |  |--Module2.bin          ' Stream: Basic module compiled code file
|  |  |--Module3.xml          ' Stream: Scrambled Basic module source file
|  |  |--Module3.bin          ' Stream: Basic module compiled code file
|  |  |--script-lb.xml        ' Stream: Basic library index file
|  |
|  |--script-lc.xml           ' Stream: Basic library container index file
|
|--<DIR> Dialogs
|  |
|  |--<DIR> Standard          ' Folder: Contains library "Standard"
|  |  |
|  |  |--dialog-lb.xml        ' Stream: Dialog library index file
|  |
|  |--<DIR> Library1          ' Folder: Contains library "Library1"
|  |  |
|  |  |--dialog-lb.xml        ' Stream: Dialog library index file
|  |
|  |--dialog-lc.xml           ' Stream: Dialog library container index file
|
|  ' From here the folders and streams have nothing to do with libraries
```

```
|--<DIR> Meta-Inf
|--content.xml
|--settings.xml
|--styles.xml
```

This example also shows that a *Dialogs* folder is created in the document package file although the library Standard and the library Library1 do not contain dialogs. This is done because the Dialog library Library1 would be lost after reloading the document. Only a single empty library Standard is assumed to exist, even if it is not stored explicitly.

11.8 Library Deployment

OpenOffice.org has a simple concept to add Basic libraries to an existing installation. Bringing Basic libraries into a OpenOffice.org installation involves the following steps:

- Package your libraries.
- Place the package into a specific package directory. There is a directory for shared packages in a network installation and a directory for user packages. This is described later.
- Close all instances of OpenOffice.org, launch a comman-line shell, change to *<OfficePath>/program* and run the tool *pkgchk* from the program directory. The tool *pkgchk* is part of the StarOffice Development Kit (SDK).

  ```
  [<OfficePath>/program] $ pkgchk my_package.zip
  ```

  The tool analyzes the packages in the package directories and matches them with a cache directory for user-defined extensions used by OpenOffice.org. Additionally, you can specify packages as command-line arguments that are copied into the package directory in advance.

The opposite steps are necessary to remove a package from your OpenOffice.org installation:

- Remove the package from the packages directory.
- Close all instances of OpenOffice.org and run *pkgchk*.

You can run *pkgchk* with the option '--help' or '-h' to get a comprehensive overview of all the switches.



Be careful not to run the *pkgchk* deployment tool while there are running instances of OpenOffice.org. For ordinary users, this case is recognized by the *pkgchk* process and leads to abortion, b is not reognized for shared network installationsusing option '--shared' or '-s'. If any user of a network installation has open processes, data inconsistencies may occur and OpenOffice.org processes may crash.

Package Structure

A UNO package is a zip file containing Basic libraries, or UNO components and type libraries. The *pkgchk* tool unzips all the packages found in the package directory into the cache directory, preserving the file structure of the zip file.

After the cache directory is ready, *pkgchk* traverses the cache directory recursively. Depending on the extension of the files it detects, it carries out the necessary registration steps. Unknown file types are ignored.

Basic libraries

The pkgchk tool links Basic library files (*.xlb*) into OpenOffice.org by adding them to the Basic library container files (*.xlc*) that reside in the following paths:

Library File	User Installation	Shared Installation
script.xlb	<OfficePath>/user/basic/script.xlc	<OfficePath>/share/basic/script.xlc
dialog.xlb	<OfficePath>/user/basic/dialog.xlc	<OfficePath>/share/basic/dialog.xlc

The files share/basic/*.xlc are created when new libraries are shared among all users using the *pkgchk* option -s (--shared) in a network installation.

The name of a Basic library is determined by the name of its parent directory. Therefore, package complete library folders, including the parent folders into the UNO Basic package. For example, if your library is named `MyLib`, there has to be a corresponding folder */ MyLib* in your development environment. This folder must be packaged completely into the UNO package, so that the zip file contains a structure similar to the following:

```
my_package.zip:
    MyLib/
        script.xlb
        dialog.xlb
        Module1.xba
        Dialog1.xba
```

Other package components

Pkgchk automatically registers shared libraries, Java archives and type libraries found in a UNO package. For details, see *4.9.1 Writing UNO Components - Deployment Options for Components - UNO Package Installation*



The autopilot *.xlb* libraries are registered in the *user/basic/*.xlc* files, but located in *share/basic*. This makes it is possible to delete and disable the autopilots for certain users even in a network installation. This is impossible for libraries deployed with the *pkgchk* tooland libraries deployed with the share option are always shared among all users.

Path Settings

The package directories are called *uno-packages* by default. There can be one in *<OfficePath>/share* for shared installations and another one in *<OfficePath>/user* for single users. The cache directories are created automatically within the respective *uno-packages* directory. OpenOffice.org has to be configured to look for these paths in the *uno.ini* file (on Windows, *unorc* on Unix) in *<OfficePath>/program*. When *pkgchk* is launched, it checks this file for package entries. If they do not exist, the following default values are added to *uno(.ini|rc)*.

```
[Bootstrap]
UNO_SHARED_PACKAGES=${$SYSBINDIR/bootstrap.ini::BaseInstallation}/share/uno_packages
UNO_SHARED_PACKAGES_CACHE=$UNO_SHARED_PACKAGES/cache
UNO_USER_PACKAGES=${$SYSBINDIR/bootstrap.ini::UserInstallation}/user/uno_packages
UNO_USER_PACKAGES_CACHE=$UNO_USER_PACKAGES/cache
```

The settings reflect the default values for the *shared* package and cache directory, and the *user* package and cache directory as described above.

In a network installation, all users start the office from a common directory on a file server. The administrator puts the packages for all the users of the network installation into the *<OfficePath>/share/uno_packages* folder of the shared installation. If a user wants to install packages locally so that only a single installation is affected, the user must copy the packages to *<OfficePath>/user/uno_packages*.

Pkgchk has to be run differently for a shared and a user installation. To install shared packages, run *pkgchk* with the -s (-shared) option which causes *pkgchk* to process only the shared packages. If *pkgchk* is run without command- line parameters, the user packages will be registered.

Additional Options

By default, the tool logs all actions into the *<cache-dir>/log.txt* file. You can switch to another log file through the -l (–log) <file name> option. Option -v (–verbose) logs to stdout, in addition to the log file.

The tool handles errors loosely. It continues after errors even if a package cannot be inflated or a shared library cannot be registered. The tool logs these errors and proceeds silently. If you want the tool to stop on every error, switch on the –strict_error handling.

If there is some inconsistency with the cache and you want to renew it from the ground up, repeating the installation using the option -r (–renewal).

12 Database Access

12.1 Overview

12.1.1 Capabilities

Platform Independence

The goal of the OpenOffice.org API database integration is to provide platform independent database connectivity for OpenOffice.org API. Well it is necessary to access database abstraction layers, such as JDBC and ODBC, it is also desirable to have direct access to arbitrary data sources, if required.

The OpenOffice.org API database integration reaches this goal through an abstraction above the abstractions with the Star Database Connectivity (SDBC). SDBC accesses data through SDBC drivers. Each SDBC driver knows how to get data from a particular source. Some drivers handle files themselves, others use a standard driver model, or existing drivers to retrieve data. The concept makes it possible to integrate database connectivity for MAPI address books, LDAP directories and OpenOffice.org Calc into the current version of OpenOffice.org API.

Since SDBC drivers are UNO components, it is possible to write drivers for data sources and thus extend the database connectivity of OpenOffice.org API.

Functioning of the OpenOffice.org API Database Integration

The OpenOffice.org API database integration is based on SQL. This section discusses how the OpenOffice.org API handles various SQL dialects and how it integrates with data sources that do not understand SQL.

OpenOffice.org API has a built-in parser that tests and adjusts the syntax to be standard SQL. With the parser, differences between SQL dialects, such as case sensitivity, can be handled if the query composer is used. Data sources that do not understand SQL can be treated by an SDBC driver that is a database engine of its own, which translates from standard SQL to the mechanisms needed to read and write data using a non-SQL data source.

Integration with OpenOffice.org API

OpenOffice.org API employs SDBC data sources in Writer, Calc and Database Forms. In Writer, use form letter fields to access database tables, create email form letters, and drag tables and queries into a document to create tables or lists.

If a table is dragged into a Calc spreadsheet, the database range that can be updated from the database, and data pilots can be created from database connections. Conversely, drag a spreadsheet range onto a database to import the spreadsheet data into a database.

Another area of database connectivity are database forms. Form controls can be inserted into Writer or Calc documents to hook them up to database tables to get data aware forms.

While there is no API coverage for direct database integration in Writer, the database connectivity in Calc and Database Forms can be controlled through the API. Refer to the corresponding chapters *8.3.5 Spreadsheet Documents - Working with Spreadsheets - Database Operations* and *13 Forms* for more information. In Writer, database connectivity can be implemented by application programmers, for example, by accessing text field context. No API exists for merging complete selections into text.

Using the OpenOffice.org API database integration enhances or automates the out-of-box database integration, creates customized office documents from databases, or provides simple, platform-independent database clients in the OpenOffice.org API environment.

12.1.2 Architecture

The OpenOffice.org API database integration is divided into three layers: SDBC, SDBCX, and SDB. Each layer extends the functionality of the layer below.

- Star Database (SDB) is the highest layer. This layer provides an application-centered view of the databases. Services, such as the database context, data sources, advanced connections, persistent query definitions and command definitions, as well as authentication and row sets are in this layer.
- Star Database Connectivity Extension (SDBCX) is the middle layer which introduces abstractions, such as catalogs, tables, views, groups, users, columns, indexes, and keys, as well as the corresponding containers for these objects.
- Star Database Connectivity (SDBC) is the lowest layer. This layer contains the basic database functionality used by the higher layers, such as drivers, simple connections, statements and result sets.

12.1.3 Example: Querying the Bibliography Database

The following example queries the bibliography database that is delivered with the OpenOffice.org distribution. The basic steps are:

1. Create a `com.sun.star.sdb.RowSet`.
2. Configure `com.sun.star.sdb.RowSet` to select from the table "biblio" in the data source "Bibliography".
3. Execute it.
4. Iterate over its rows.

5. Insert a new row.

If the database requires login, set additional properties for user and password, or connect using interactive login. There are other options as well. For details, refer to the section *12.3.1 Database Access - Manipulating Data - The RowSet Service*. (Database/OpenQuery.java)

```
protected void openQuery() throws com.sun.star.uno.Exception, java.lang.Exception {
    xRemoteServiceManager = this.getRemoteServiceManager(
        "uno:socket,host=localhost,port=8100;urp;StarOffice.ServiceManager");

    // first we create our RowSet object and get its XRowSet interface
    Object rowSet = xRemoteServiceManager.createInstanceWithContext(
       "com.sun.star.sdb.RowSet", xRemoteContext);

    com.sun.star.sdbc.XRowSet xRowSet = (com.sun.star.sdbc.XRowSet)
        UnoRuntime.queryInterface(com.sun.star.sdbc.XRowSet.class, rowSet);

    // set the properties needed to connect to a database
    XPropertySet xProp = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, xRowSet);

    // the DataSourceName can be a data source registered with OpenOffice.org, among other possibilities
    xProp.setPropertyValue("DataSourceName", "Bibliography");

    // the CommandType must be TABLE, QUERY or COMMAND - here we use COMMAND
    xProp.setPropertyValue("CommandType", new Integer(com.sun.star.sdb.CommandType.COMMAND));

    // the Command could be a table or query name or a SQL command, depending on the CommandType
    xProp.setPropertyValue("Command", "SELECT IDENTIFIER, AUTHOR FROM biblio");

    // if your database requires logon, you can use the properties User and Password
    // xProp.setPropertyValue("User", "JohnDoe");
    // xProp.setPropertyValue("Password", "mysecret");

    xRowSet.execute();

    // prepare the XRow and XColumnLocate interface for column access
    // XRow gets column values
    com.sun.star.sdbc.XRow xRow = (com.sun.star.sdbc.XRow)UnoRuntime.queryInterface(
        com.sun.star.sdbc.XRow.class, xRowSet);
    // XColumnLocate finds columns by name
    com.sun.star.sdbc.XColumnLocate xLoc = (com.sun.star.sdbc.XColumnLocate)UnoRuntime.queryInterface(
        com.sun.star.sdbc.XColumnLocate.class, xRowSet);

    // print output header
    System.out.println("Identifier\tAuthor");
    System.out.println("----------\t------");

    // output result rows
    while ( xRowSet.next() ) {
        String ident = xRow.getString(xLoc.findColumn("IDENTIFIER"));
        String author = xRow.getString(xLoc.findColumn("AUTHOR"));
        System.out.println(ident + "\t\t" + author);
    }

    // insert a new row
    // XResultSetUpdate for insertRow handling
    com.sun.star.sdbc.XResultSetUpdate xResultSetUpdate = (com.sun.star.sdbc.XResultSetUpdate)
        UnoRuntime.queryInterface(
            com.sun.star.sdbc.XResultSetUpdate.class, xRowSet);

    // XRowUpdate for row updates
    com.sun.star.sdbc.XRowUpdate xRowUpdate = (com.sun.star.sdbc.XRowUpdate)
        UnoRuntime.queryInterface(
            com.sun.star.sdbc.XRowUpdate.class, xRowSet);

    // move to insertRow buffer
    xResultSetUpdate.moveToInsertRow();

    // edit insertRow buffer
    xRowUpdate.updateString(xLoc.findColumn("IDENTIFIER"), "GOF95");
    xRowUpdate.updateString(xLoc.findColumn("AUTHOR"), "Gamma, Helm, Johnson, Vlissides");

    // write buffer to database
    xResultSetUpdate.insertRow();

    // throw away the row set
    com.sun.star.lang.XComponent xComp = (com.sun.star.lang.XComponent)UnoRuntime.queryInterface(
        com.sun.star.lang.XComponent.class, xRowSet);
    xComp.dispose();
}
```

12.2 Data Sources in OpenOffice.org API

12.2.1 DatabaseContext

In the OpenOffice.org graphical user interface (GUI), define data sources using the data source administrator and access them in the database browser. A data source has four main aspects. It contains:

- The *general information* necessary to connect to a data source.
- Settings to control the presentation of *tables*.
- SQL *query definitions*.
- *Links* to OpenOffice.org API documents, primarily documents containing database forms.

From the API perspective, these functions are mirrored in the `com.sun.star.sdb.DatabaseContext` service. The database context is a container for data sources. It is a singleton, that is, it may exist only once in a running OpenOffice.org API instance and can be accessed by creating it at the global service manager of the office.

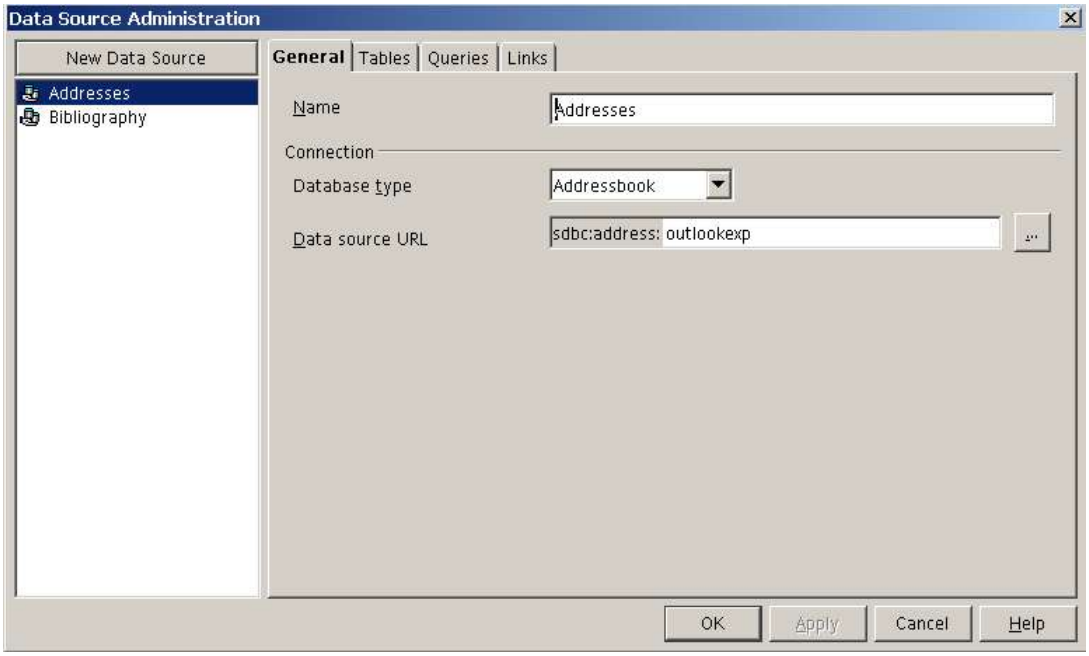


Illustration 170: The Dialog "Data Source Administration"

The database context is the entry point for applications that need to connect to a data source already defined in the OpenOffice.org API. Additionally, it is used to create new data sources and add them to OpenOffice.org API. The following figure shows the relationship between the database context, the data sources and the connection over a data source.

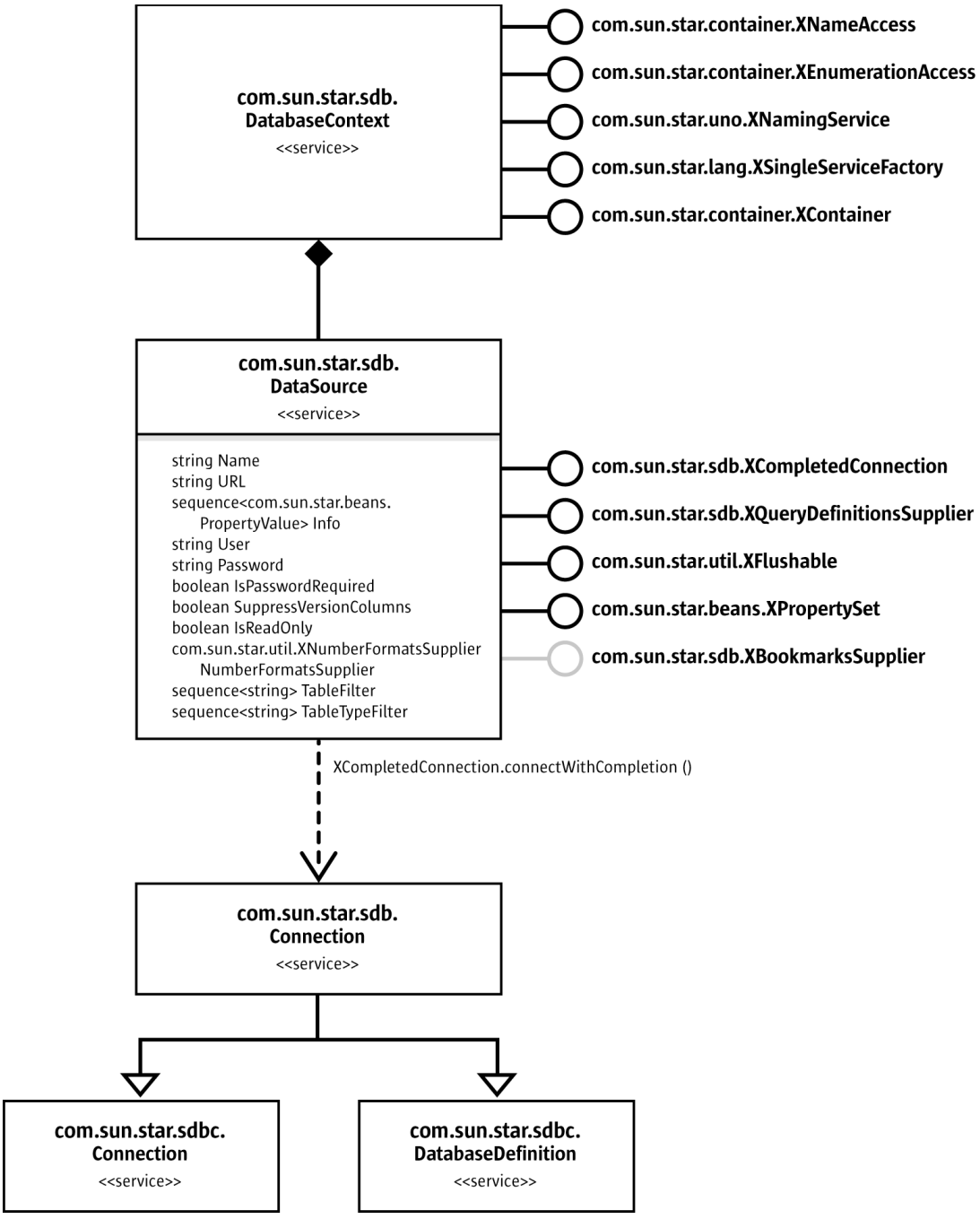


Illustration 171: `com.sun.star.sdb.DatabaseContext`

The database context is used to get a data source that provides a `com.sun.star.sdb.Connection` through its `com.sun.star.sdb.XCompletedConnection` interface.

Existing data sources are obtained from the database context at its interfaces `com.sun.star.container.XNameAccess` and `com.sun.star.container.XEnumeration`. Their methods `getByName()` and `createEnumeration()` deliver the `com.sun.star.sdb.DataSource` services defined in the OpenOffice.org GUI.

The code below shows how to print all available data sources: (Database/CodeSamples.java)

```
// prints all data sources
public static void printDataSources(XMultiServiceFactory _rMSF) throws com.sun.star.uno.Exception {
```

```
    // retrieve the DatabaseContext and get its com.sun.star.container.XNameAccess interface
    XNameAccess xNameAccess = (XNameAccess)UnoRuntime.queryInterface(
        XNameAccess.class, _rMSF.createInstance("com.sun.star.sdb.DatabaseContext"));

    // print all DataSource names
    String aNames [] = xNameAccess.getElementNames();
    for (int i=0;i<aNames.length;++i)
        System.out.println(aNames[i]);
}
```

12.2.2 DataSources

The DataSource Service

The `com.sun.star.sdb.DataSource` service includes all the features of a database defined in OpenOffice.org API. `DataSource` provides the following properties for its knowledge about how to connect to a database and which tables to display:

Properties of `com.sun.star.sdb.DataSource`	
`Name`	`[readonly] string` — The name of the data source.
`URL`	`string` — Indicates a database URL. Valid URL formats are: `jdbc: subprotocol : subname` `sdbc: subprotocol : subname`
`Info`	`sequence< com.sun.star.beans.PropertyValue >`. A list of arbitrary `string` tag or value pairs as connection arguments.
`User`	`String` — The login name of the current user.
`Password`	`string` — The password of the current user. It is not stored with the data source.
`IsPasswordRequired`	`boolean` — Indicates that a password is always necessary and might be interactively requested from the user by an interaction handler.
`IsReadOnly`	`[readonly] boolean` — Determines if database contents may be modified.
`NumberFormatsSupplier`	`[readonly]` [idl.com.sun.star.util.XNumberFormatsSupplier]. Provides an object for number formatting.
`TableFilter`	`sequence< string >`. A list of tables the data source should display. If empty, all tables are hidden. Valid placeholders are % and ?.
`TableTypeFilter`	`sequence< string >`. A list of table types the DataSource should display. If empty, all table types are rejected. Possible type strings are TABLE, VIEW, and SYSTEM TABLE.
`SuppressVersionColumns`	`boolean` — Indicates that components displaying data obtained from this data source should suppress columns used for versioning.

All other capabilities of a `DataSource`, such as query definitions, document links, and the actual process of establishing connections and flushing pending configuration changes are available over its interfaces.

- `com.sun.star.sdb.XQueryDefinitionsSupplier` provides access to SQL query definitions for a database. The definition of queries is discussed in the next section, *12.2.2 Database Access - Data Sources in OpenOffice.org API - DataSources - Queries*.
- `com.sun.star.sdb.XCompletedConnection` connects to a database. It asks the user to supply necessary information before it connects. The section *12.2.3 Database Access - Data Sources in*

OpenOffice.org API - Connections - Connecting Through a DataSource shows how to establish a connection.

- `com.sun.star.sdb.XBookmarksSupplier` provides access to bookmarks pointing at documents associated with the DataSource, primarily OpenOffice.org API documents containing form components. Although it is optional, it is implemented for all data sources in OpenOffice.org API. The section *12.2.2 Database Access - Data Sources in OpenOffice.org API - DataSources - Forms and Other Links* explains database bookmarks..
- `com.sun.star.util.XFlushable` forces the data source to flush all information including the properties above to the configuration repository. However, changes work immediately and are stored in the OpenOffice.org configuration.

Adding and Editing Datasources

New data sources have to be created by the `com.sun.star.lang.XSingleServiceFactory` interface of the database context. A new data source must be registered with the database context at its `com.sun.star.uno.XNamingService` interface and the necessary properties set.

The lifetime of data sources is controlled through the interfaces `com.sun.star.lang.XSingleServiceFactory`, `com.sun.star.uno.XNamingService` and `com.sun.star.container.XContainer` of the database context.

The method `createInstance()` of XSingleServiceFactory creates new generic data sources. They are added to the database context using `registerObject()` at the interface `com.sun.star.uno.XNamingService`. The `XNamingService` allows registering data sources, as well as revoking the registration. The following are the methods defined for `XNamingService`:

```
void registerObject( [in] string Name, [in] com::sun::star::uno::XInterface Object)
void revokeObject( [in] string Name)
com::sun::star::uno::XInterface getRegisteredObject( [in] string Name)
```

In the following example, a data source is created for a previously generated Adabas D database named MYDB1 on the local machine. The `URL` property has to be present, and for Adabas D the property `IsPasswordRequired` should be `true`, otherwise no interactive connection can be established. The password dialog requests a user name by setting the `User` property. (Database/CodeSamples.java)

```
// creates a new DataSource
public static void createNewDataSource(XMultiServiceFactory _rMSF) throws com.sun.star.uno.Exception {
    // the XSingleServiceFactory of the database context creates new generic
    // com.sun.star.sdb.DataSources (!)
    // retrieve the database context at the global service manager and get its
    // XSingleServiceFactory interface
    XSingleServiceFactory xFac = (XSingleServiceFactory)UnoRuntime.queryInterface(
        XSingleServiceFactory.class, _rMSF.createInstance("com.sun.star.sdb.DatabaseContext"));

    // instantiate an empty data source at the XSingleServiceFactory
    // interface of the DatabaseContext
    Object xDs = xFac.createInstance();

    // register it with the database context
    XNamingService xServ = (XNamingService)UnoRuntime.queryInterface(XNamingService.class, xFac);
    xServ.registerObject("NewDataSourceName", xDs);

    // setting the necessary data source properties
    XPropertySet xDsProps = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, xDs);
    // Adabas D URL
    xDsProps.setPropertyValue("URL", "sdbc:adabas::MYDB1");

    // force password dialog
    xDsProps.setPropertyValue("IsPasswordRequired", new Boolean(true));

    // suggest dsadmin as user name
    xDsProps.setPropertyValue("User", "dsadmin");
}
```

The various possible database URLs are discussed in the section *12.2.3 Database Access - Data Sources in OpenOffice.org API - Connections - Driver Specifics*.

To edit an existing data source, retrieve it by name from the `com.sun.star.container.XNameAccess` interface of the database context and use its `com.sun.star.beans.XPropertySet` interface to configure it, as required.

Queries

A `com.sun.star.sdb.QueryDefinition` encapsulates a definition of an SQL statement stored in OpenOffice.org API. It is similar to a view or a stored procedure, because it can be reused, and executed and altered by the user in the GUI. It is possible to run a `QueryDefinition` against a different database by changing the underlying `DataSource` properties. It can also be created without being connected to a database.

The purpose of the query services available at a `DataSource` is to define and edit queries. The query services by themselves do not offer methods to execute queries. To open a query, use a `com.sun.star.sdb.RowSet` service or the `com.sun.star.sdb.XCommandPreparation` interface of a connection. See the sections *12.3.1 Database Access - Manipulating Data - The RowSet Service* and *12.3.6 Database Access - Manipulating Data - PreparedStatement From DataSource Queries* for additional details.

Adding and Editing Predefined Queries

The query definitions container `com.sun.star.sdb.DefinitionContainer` is used to work with the query definitions of a data source. It is returned by the `com.sun.star.sdb.XQueryDefinitionsSupplier` interface of the data source, which has a single method for this purpose:

```
com::sun::star::container::XNameAccess getQueryDefinitions()
```

The `DefinitionContainer` is not only an `XNameAccess`, but a `com.sun.star.container.XNameContainer`, that is, add new query definitions by name (see *2 First Steps*). Besides the name access, obtain query definitions through `com.sun.star.container.XIndexAccess` and `com.sun.star.container.XEnumerationAccess`.

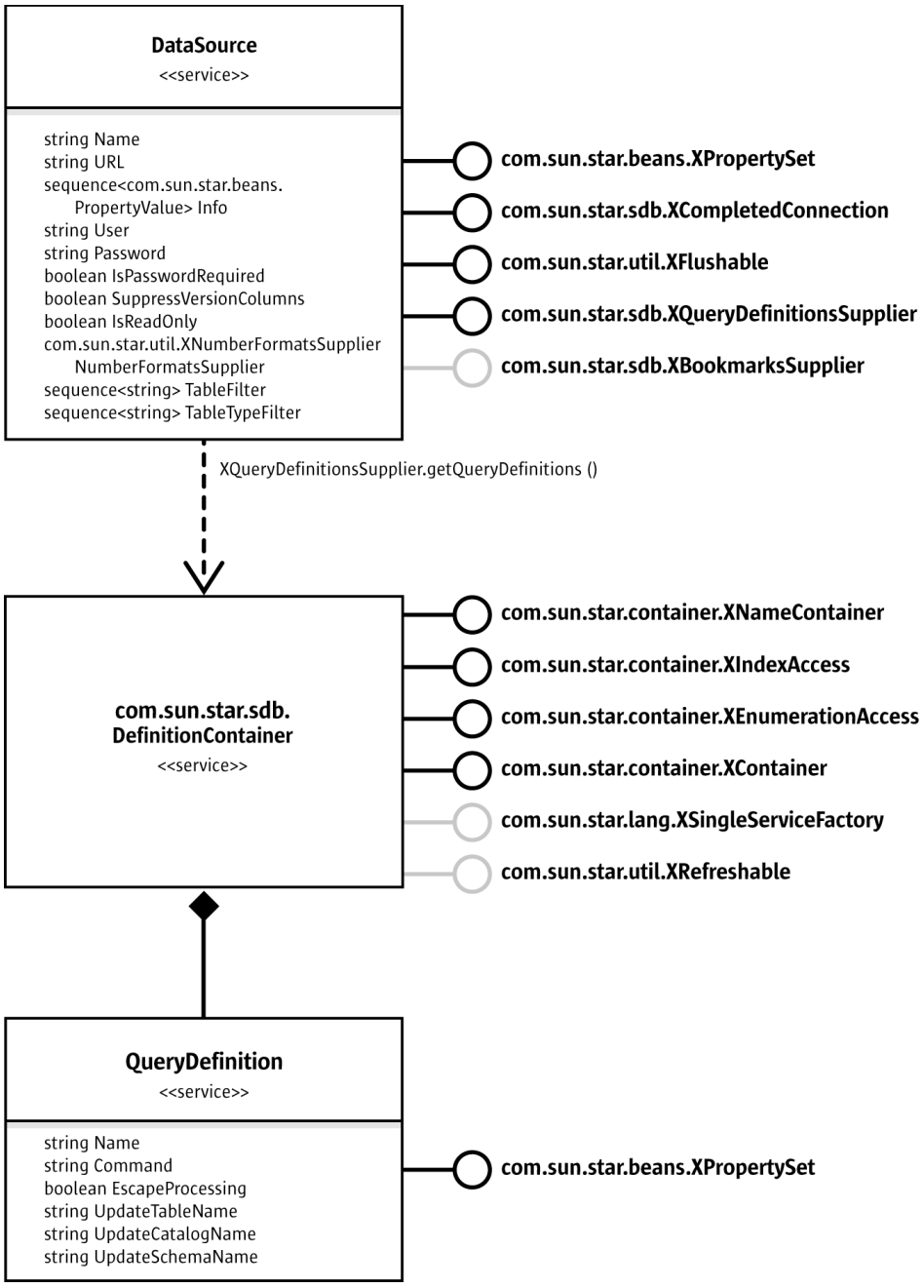


Illustration 172: DefinitionContainer And QueryDefinition

New query definitions are created by the `com.sun.star.lang.XSingleServiceFactory` interface of the query definitions container. Its method `createInstance()` provides an empty `QueryDefinition` to configure, as required. Then, the new query definition is added to the `DefinitionContainer` using `insertByName()` at the `XNameContainer` interface.



The optional interface `com.sun.star.util.XRefreshable` is not supported by the DefinitionContainer implementation.

A `QueryDefinition` is configured through the following properties:

Properties of `com.sun.star.sdb.QueryDefinition`	
`Name`	`string` — The name of the queryDefinition.
`Command`	`string` — The SQL SELECT command.
`EscapeProcessing`	`boolean` — If `true`, determines that the query must not be touched by the built-in SQL parser of OpenOffice.org API.
`UpdateCata-logName`	`string` — The name of the update table catalog used to identify tables, supported by some databases.
`UpdateSchemaName`	`string` — The name of the update table schema used to identify tables, supported by some databases.
`UpdateTableName`	`string` The name of the update table catalog used to identify tables, supported by some databases The name of the table which should be updated. This is usually used for queries based on more than one table and makes such queries partially editable. The property `UpdateTableName` must contain the name of the table with unique rows in the result set. In a 1:n join this is usually the table on the n side of the join.

The following example adds a new query definition `Query1` to the data source Bibliography that is provided with OpenOffice.org API. (Database/CodeSamples.java)

```
// creates a new query definition named Query1
public static void createQuerydefinition(XMultiServiceFactory _rMSF) throws com.sun.star.uno.Exception {
    XNameAccess xNameAccess = (XNameAccess) UnoRuntime.queryInterface(
        XNameAccess.class, _rMSF.createInstance( "com.sun.star.sdb.DatabaseContext") );

    // we use the datasource Bibliography
    XQueryDefinitionsSupplier xQuerySup = (XQueryDefinitionsSupplier) UnoRuntime.queryInterface(
        XQueryDefinitionsSupplier.class, xNameAccess.getByName( "Bibliography" ));

    // get the container for query definitions
    XNameAccess xQDefs = xQuerySup.getQueryDefinitions();

    // for new query definitions we need the com.sun.star.lang.XSingleServiceFactory interface
    // of the query definitions container
    XSingleServiceFactory xSingleFac = (XSingleServiceFactory)UnoRuntime.queryInterface(
        XSingleServiceFactory.class, xQDefs);

    // order a new query and get its com.sun.star.beans.XPropertySet interface
    XPropertySet xProp = (XPropertySet) UnoRuntime.queryInterface(
        XPropertySet.class, xSingleFac.createInstance());

    // configure the query
    xProp.setPropertyValue("Command","SELECT * FROM biblio");
    xProp.setPropertyValue("EscapeProcessing", new Boolean(true));

    // insert it into the query definitions container
    XNameContainer xCont = (XNameContainer) UnoRuntime.queryInterface(
        XNameContainer.class, xQDefs);

    xCont.insertByName("Query1", xProp);
}
```

Runtime Settings For Predefined Queries

The queries in the user interface have a number of advanced settings concerning the formatting and filtering of the query and its columns. For the API, these settings are available as long as the data source is connected with the underlying database. The section *12.2.3 Database Access - Data Sources in OpenOffice.org API - Connections - Connecting Through a DataSource* discusses how to get a connection from a data source. When the connection is made, its interface `com.sun.star.sdb.XQueriesSupplier` returns query objects with the advanced settings above.

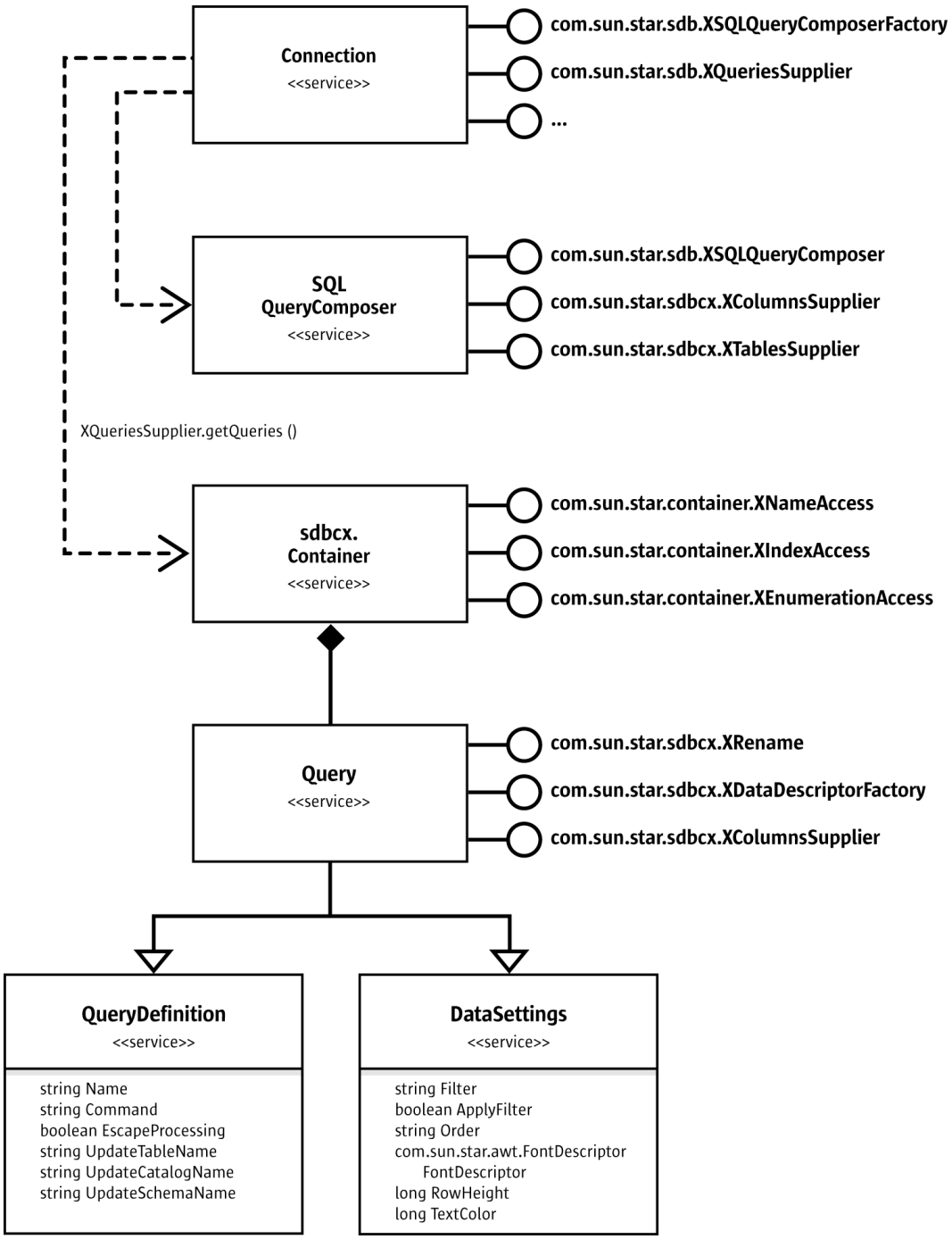


Illustration 173: Connection, QueryComposer And Query in the sdb Module

The `Connection` gives you a `com.sun.star.sdbcx.Container` of `com.sun.star.sdb.Query` services. These `Query` objects are different from `QueryDefinitions`.

The `com.sun.star.sdb.Query` service inherits both the properties from `com.sun.star.sdb.QueryDefinition` service described previously, and the properties defined in the service `com.sun.star.sdb.DataSettings`. Use `DataSettings` to customize the appearance of the query when used in the OpenOffice.org API GUI or together with a `com.sun.star.sdb.RowSet`.

Properties of com.sun.star.sdb.DataSettings	
Filter	string — An additional filter for the data object, WHERE clause syntax.
ApplyFilter	boolean — Indicates if the filter should be applied, default is FALSE.
Order	string — Is an additional sort order definition.
FontDescriptor	struct com.sun.star.awt.FontDescriptor. Specifies the font attributes for displayed data.
RowHeight	long — Specifies the height of a data row.
TextColor	long — Specifies the text color for displayed text in 0xAARRGGBB notation

In addition to these properties, the com.sun.star.sdb.Query service offers a com.sun.star.sdbcx.XDataDescriptorFactory to create new query descriptors based on the current query information. Use this query descriptor to append new queries to the com.sun.star.sdbcx.Container using its com.sun.star.sdbcx.XAppend interface. This is an alternative to the connection-independent method to create new queries as discussed above. The section *12.4.3 Database Access - Database Design - Using SDBCX to Access the Database Design - The Descriptor Pattern* explains how to use descriptors to append new elements to database objects.

The com.sun.star.sdbcx.XRename interface is used to rename a query. It has one method:

```
void rename( [in] string newName)
```

The interface com.sun.star.sdbcx.XColumnsSupplier grants access to the column settings of the query through its single method getColumns():

```
com::sun::star::container::XNameAccess getColumns()
```

The columns returned by getColumns() are com.sun.star.sdb.Column services that provide column information and the ability to improve the appearance of columns. This service is explained in the section *12.2.2 Database Access - Data Sources in OpenOffice.org API - DataSources - Tables and Columns*.

The following code sample connects to Bibliography, and prints the column names and types of the previously defined query Query1. (Database/CodeSamples.java)

```
public static void printQueryColumnNames(XMultiServiceFactory _rMSF) throws com.sun.star.uno.Exception {
    XNameAccess xNameAccess = (XNameAccess)UnoRuntime.queryInterface(
    XNameAccess.class,_rMSF.createInstance("com.sun.star.sdb.DatabaseContext"));

    // we use Bibliography
    XDataSource xDS = (XDataSource)UnoRuntime.queryInterface(
        XDataSource.class, xNameAccess.getByName("Bibliography"));

    // simple way to connect
    XConnection con = xDS.getConnection("", "");

    // we need the XQueriesSupplier interface of the connection
    XQueriesSupplier xQuerySup = (XQueriesSupplier)UnoRuntime.queryInterface(
        XQueriesSupplier.class, con);

    // get container with com.sun.star.sdb.Query services
    XNameAccess xQDefs = xQuerySup.getQueries();

    // retrieve XColumnsSupplier of Query1
    XColumnsSupplier xColsSup = (XColumnsSupplier) UnoRuntime.queryInterface(
        XColumnsSupplier.class,xQDefs.getByName("Query1"));

    XNameAccess xCols = xColsSup.getColumns();

    // Access column property TypeName
    String aNames [] = xCols.getElementNames();
    for (int i=0;i<aNames.length;++i) {
        Object col = xCols.getByName(aNames[i]);
        XPropertySet xColumnProps = (XPropertySet)UnoRuntime.queryInterface(
            XPropertySet.class, col);
        System.out.println(aNames[i] + " " + xColumnProps.getPropertyValue("TypeName"));
    }
}
```

The SQLQueryComposer

The service `com.sun.star.sdb.SQLQueryComposer` is a tool that composes SQL `SELECT` strings. It hides the complexity of parsing and evaluating SQL statements, and provides sophisticated methods to configure an SQL statement with filtering and ordering criteria. A query composer is retrieved over the `com.sun.star.sdb.XSQLQueryComposerFactory` interface of a `com.sun.star.sdb.Connection`:

```
com::sun::star::sdb::XSQLQueryComposer createQueryComposer()
```

Its interface `com.sun.star.sdb.XSQLQueryComposer` is used to supply the SQLQueryComposer with the necessary information. It has the following methods:

```
// provide SQL string
void setQuery( [in] string command)
string getQuery()
string getComposedQuery()

// control the WHERE clause
void setFilter( [in] string filter)
void appendFilterByColumn( [in] com::sun::star::beans::XPropertySet column)
string getFilter()
sequence< sequence< com::sun::star::beans::PropertyValue > > getStructuredFilter()

// control the ORDER BY clause
void setOrder( [in] string order)
void appendOrderByColumn( [in] com::sun::star::beans::XPropertySet column, [in] boolean ascending)
string getOrder()
```

In the above method, a query command, such as "SELECT Identifier, Address, Author FROM biblio" is passed to `setQuery()`, then the criteria for `WHERE` and `ORDER BY` is added. The `WHERE` expressions are passed without the `WHERE` keyword to `setFilter()`, and the method `setOrder()` with comma-separated `ORDER BY` columns or column numbers is provided.

As an alternative, add `WHERE` conditions using `appendFilterByColumn()`. This method expects a `com.sun.star.sdb.DataColumn` service providing the name and the value for the filter. Similarly, the method `appendOrderByColumn()` adds columns that are used for ordering. These columns could come from the `RowSet`.

Retrieve the resulting SQL string from `getComposedQuery()`.

The methods `getQuery()`, `getFilter()` and `getOrder()` return the `SELECT`, `WHERE` and `ORDER BY` part of the SQL command as a string.

The method `getStructuredFilter()` returns the filter split into OR levels. Within each OR level, filters are provided as AND criteria with the name of the column and the filter condition string.

The following example prints the structured filter.

```
// prints the structured filter
public static void printStructeredFilter(XMultiServiceFactory _rMSF) throws com.sun.star.uno.Exception {
    XNameAccess xNameAccess = (XNameAccess)UnoRuntime.queryInterface(
        XNameAccess.class, _rMSF.createInstance("com.sun.star.sdb.DatabaseContext"));
    // we use the first datasource
    XDataSource xDS = (XDataSource)UnoRuntime.queryInterface(
        XDataSource.class, xNameAccess.getByName("Bibliography"));
    XConnection con = xDS.getConnection("", "");
    XQueriesSupplier xQuerySup = (XQueriesSupplier)UnoRuntime.queryInterface(
        XQueriesSupplier.class, con);

    XNameAccess xQDefs = xQuerySup.getQueries();

    XPropertySet xQuery = (XPropertySet) UnoRuntime.queryInterface(
        XPropertySet.class,xQDefs.getByName("Query1"));
    String sCommand = (String)xQuery.getPropertyValue("Command");

    XSQLQueryComposerFactory xQueryFac = (XSQLQueryComposerFactory) UnoRuntime.queryInterface(
        XSQLQueryComposerFactory.class, con);

    XSQLQueryComposer xQComposer = xQueryFac.createQueryComposer();
    xQComposer.setQuery(sCommand);

    PropertyValue aFilter [][] = xQComposer.getStructuredFilter();
    for (int i=0; i<aFilter.length; ) {
```

```
        System.out.println("( ");
        for (int j=0; j<aFilter[i].length; ++j)
            System.out.println("Name: " + aFilter[i][j].Name + " Value: " + aFilter[i][j].Value);
            System.out.println(")");
            ++i;
            if (i<aFilter.length )
                System.out.println(" OR ");
        }
    }
}
```

The interface `com.sun.star.sdbcx.XTablesSupplier` provides access to the tables that are used in the "FROM" part of the SQL-Statement:

```
com::sun::star::container::XNameAccess getTables()
```

The interface `com.sun.star.sdbcx.XColumnsSupplier` provides the selected columns, which are listed after the SELECT keyword:

```
com::sun::star::container::XNameAccess getColumns()
```

Forms and Other Links

Each data source can maintain an arbitrary number of document links. The primary purpose of this function is to provide a collection of database forms used with a database. These links are available at the `com.sun.star.sdb.XBookmarksSupplier` interface of a data source that has one method:

```
com::sun::star::container::XNameAccess getBookmarks()
```

The returned service is a `com.sun.star.sdb.DefinitionContainer`. The `DefinitionContainer` is not only an `XNameAccess`, but a `com.sun.star.container.XNameContainer`, that is, new links are added using `insertByName()` as described in the chapter *2 First Steps*. Besides the name access, links are obtained through `com.sun.star.container.XIndexAccess` and `com.sun.star.container.XEnumerationAccess`.

The returned bookmarks are simple strings containing URLs. Usually forms are are stored at *file:///* URLs. The following example adds a new document to the data source Bibliography:

```
public static void addDocumentLink(XMultiServiceFactory _rMSF) throws com.sun.star.uno.Exception {
    XNameAccess xNameAccess = (XNameAccess)UnoRuntime.queryInterface(
        XNameAccess.class,_rMSF.createInstance("com.sun.star.sdb.DatabaseContext"));

    // we use the predefined Bibliography data source
    XDataSource xDS = (XDataSource)UnoRuntime.queryInterface(
        XDataSource.class, xNameAccess.getByName("Bibliography"));

    // we need the XBookmarksSupplier interface of the data source
    XBookmarksSupplier xBookmarksSupplier = ( XBookmarksSupplier )UnoRuntime.queryInterface(
        XBookmarksSupplier.class, xDS);

    // get container with bookmark URLs
    XNameAccess xBookmarks = xBookmarksSupplier.getBookmarks();
    XNameContainer xBookmarksContainer = (XNameContainer)UnoRuntime.queryInterface(
        XNameContainer.class, xBookmarks);

    // insert new link
    xBookmarksContainer.insertByName("MyLink", "file:///home/ada01/Form_Ada01_DSADMIN.Table1.sxw");
}
```

To load a linked document, use the bookmark URL with the method `loadComponentFromUrl()` at the `com.sun.star.frame.XComponentLoader` interface of the `com.sun.star.frame.Desktop` singleton that is available at the global service manager. For details about the `Desktop`, see *6 Office Development*.

Tables and Columns

A `com.sun.star.sdb.Table` encapsulates tables in a OpenOffice.org API data source. The `com.sun.star.sdb.Table` service changes the appearance of a table and its columns in the GUI, and it contains read-only information about the table definition, such as the table name and type, the schema and catalog name, and access privileges.

It is also possible to alter the table definition at the `com.sun.star.sdb.Table` service. This is discussed in the section *12.4 Database Access - Database Design* below.

The table related services in the database context are unable to access the data in a database table. Use the `com.sun.star.sdb.RowSet` service, or to establish a connection to a database and use its `com.sun.star.sdb.XCommandPreparation` interface to manipulate table data. For details, see the sections *12.3.1 Database Access - Manipulating Data - The RowSet Service* and *12.3.6 Database Access - Manipulating Data - PreparedStatement From DataSource Queries*.

The following illustration shows the relationship between the `com.sun.star.sdb.Connection` and the `Table` objects it provides, and the services included in `com.sun.star.sdb.Table`.

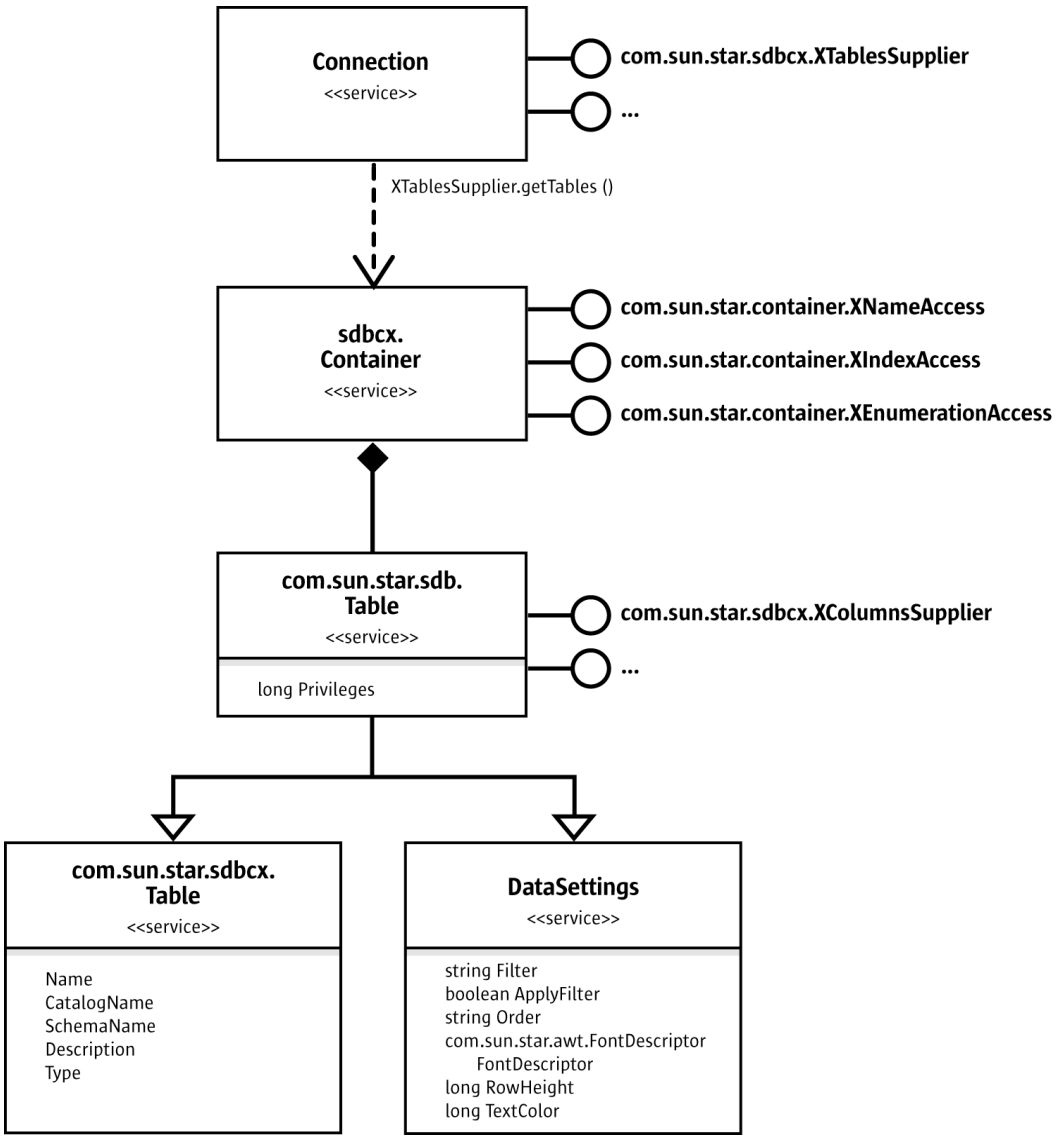


Illustration 174: Connection and Tables

The com.sun.star.sdbcx.XTablesSupplier interface of a Connection supplies a com.sun.star.sdbcx.Container of com.sun.star.sdb.Table services through its method getTables(). The container administers Table services by name, index or as enumeration.

Just like queries, tables include the display properties specified in com.sun.star.sdb.DataSettings:

Properties of com.sun.star.sdb.DataSettings	
Filter	string — An additional filter for the data object, WHERE clause syntax.
ApplyFilter	boolean — Indicates if the filter should be applied. The default is FALSE.
Order	string — Is an additional sort order definition.
FontDescriptor	Struct [idl:com.sun.star.awt.FontDescriptor]. Specifies the font attributes for displayed data.
RowHeight	long — Specifies the height of a data row.
TextColor	long — Specifies the text color for displayed text in 0xAARRGGBB notation

Basic table information is included in the properties included with com.sun.star.sdbcx.Table:

Properties of com.sun.star.sdbcx.Table	
Name	[readonly] string — Table name.
CatalogName	[readonly] string — Catalog name.
SchemaName	[readonly] string — Schema name.
Description	[readonly] string — Table Description, if supported by the driver.
Type	[readonly] string — Table type, possible values are TABLE, VIEW, SYSTEM TABLE or an empty string if the driver does not support different table types.

The service com.sun.star.sdb.Table is an extension of the service com.sun.star.sdbcx.Table. It introduces an additional property called Privileges. The Privileges property indicates the actions the current user may carry out on the table.

Properties of com.sun.star.sdb.Table	
Privileges	[readonly] long, constants group com.sun.star.sdbcx.Privilege. The property contains a bitwise AND combination of the following privileges: • SELECT user can read the data. • INSERT user can insert new data. • UPDATE user can update data. • DELETE user can delete data. • READ user can read the structure of a definition object. • CREATE user can create a definition object. • ALTER user can alter an existing object. • REFERENCE user can set foreign keys for a table. • DROP user can drop a definition object.

The appearance of single columns in a table can be changed. The following illustration depicts the service com.sun.star.sdb.Column and its relationship with the com.sun.star.sdb.Table service.

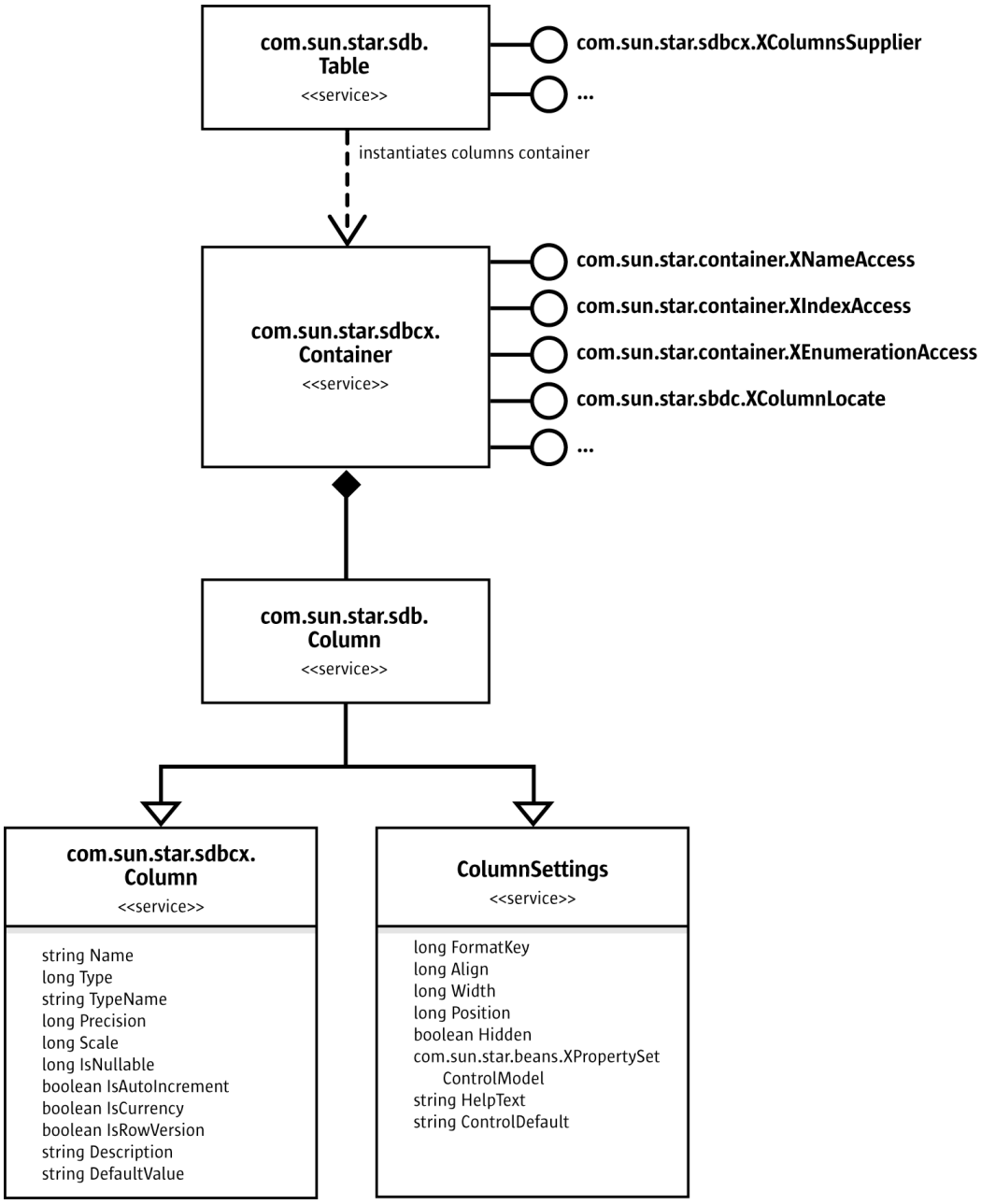


Illustration 175: Table and Table Column

For this purpose, `com.sun.star.sdb.Table` supports the interface `com.sun.star.sdbcx.XColumnsSupplier`. Its method `getColumns()` returns a `com.sun.star.sdbcx.Container` with the additional column-related interface `com.sun.star.sdbc.XColumnLocate` that is useful to get the column number for a certain column in a table:

```
long findColumn( [in] string columnName)
```

The service com.sun.star.sdb.Column combines com.sun.star.sdbcx.Column and the com.sun.star.sdb.ColumnSettings to form a column service with the opportunity to alter the visual appearance of a column.

Properties of com.sun.star.sdb.ColumnSettings	
FormatKey	long — Contains the index of the number format that is used for the column. The proper value can be determined using the com.sun.star.util.XNumberFormatter interface. If the value is void, a default number format is used according to the data type of the column.
Align	long — Specifies the alignment of column text. Possible values are: 0: left 1: center 2: right If the value is void, a default alignment is used according to the data type of the column.
Width	long — Specifies the width of the column displayed in a grid. The unit is 10th mm. If the value is void, a default width should be used according to the label of the column.
Position	long — The ordinal position of the column within a grid. If the value is void, the default position should be used according to their order of appearance in com.sun.star.sdbc.XResultSetMetaData.
Hidden	boolean — Determines if the column should be displayed.
ControlModel	com.sun.star.beans.XPropertySet. May contain a control model that defines the settings for layout. The default is NULL.
HelpText	string — Describes an optional help text that can be used by UI components when representing this column.
ControlDefault	string — Contains the default value that should be displayed by a control when moving to a new row.

The Properties of com.sun.star.sdbcx.Column are readonly and can be used for information purposes:

Properties of com.sun.star.sdbcx.Column	
Name	[readonly] string — The name of the column.
Type	[readonly] long — The [idl:com.sun.star.sdbc.DataType] of the column.
TypeName	[readonly] string — The type name used by the database. If the column type is a user-defined type, then a fully-qualified type name is returned. May be empty.
Precision	[readonly] long — The number of decimal digits or chars.
Scale	[readonly] long — Number of digits after the decimal point.
IsNullable	[readonly] long, constants group com.sun.star.sdbc.ColumnValue. Indicates if values may be NULL in the designated column. Possible values are: NULLABLE: column allows NULL values. NO_NULLS: column does not allow NULL values. NULLABLE_UNKNOWN : it is unknown whether or not NULL is allowed
IsAutoIncrement	[readonly] boolean — Indicates if the column is automatically numbered.
IsCurrency	[readonly] boolean — Indicates if the column is a cash value.

Properties of `com.sun.star.sdbcx.Column`	
`IsRowVersion`	`[readonly] boolean` — Indicates whether the column contains a type of time or date stamp used to track updates.
`Description`	`[readonly] string` — Keeps a description of the object.
`DefaultValue`	`[readonly] string` — Keeps a default value for a column, and is provided as a string.

12.2.3 Connections

Understanding Connections

A *connection* is an open communication channel to a database. A connection is required to work with data in a database or with a database definition. Connections are encapsulated in `Connection` objects in the OpenOffice.org API. There are several possibilities to get a `Connection`:

- Connect to a data source that has already been set up in the database context of OpenOffice.org API.
- Use the driver manager or a specific driver to connect to a database without using an existing data source from the database context.
- Get a connection from the connection pool maintained by OpenOffice.org API.
- Reuse the connection of a database form which is currently open in the GUI.

With the above possibilities, a `com.sun.star.sdb.Connection` is made or at least a `com.sun.star.sdbc.Connection`:

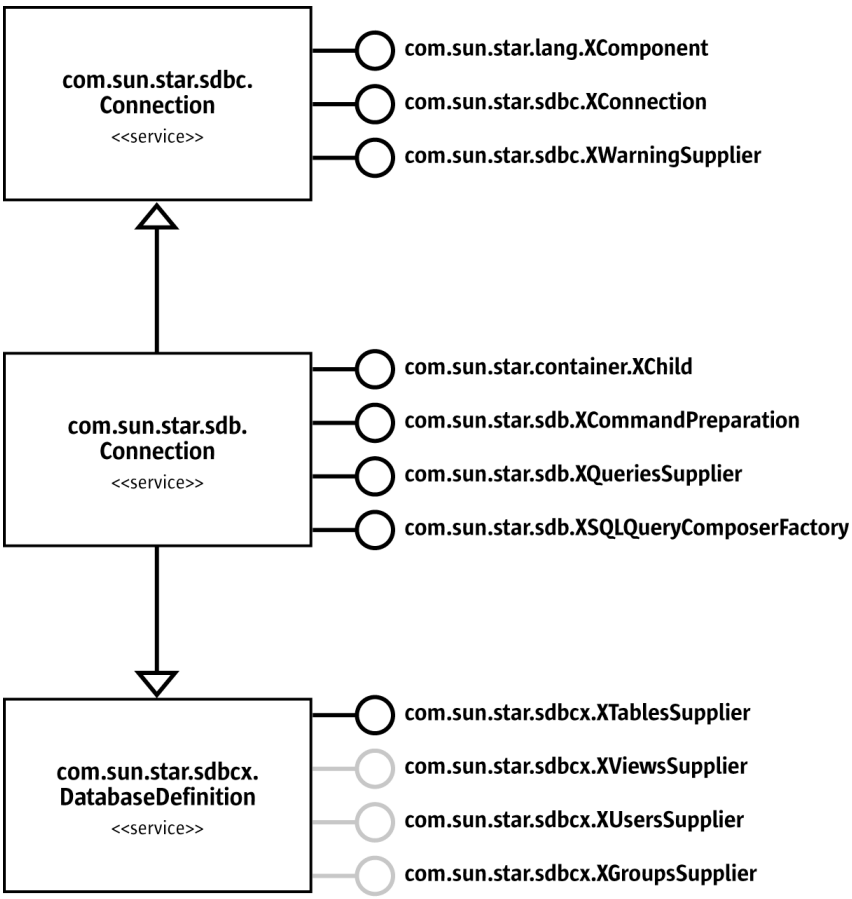


Illustration 176: `com.sun.star.sdb.Connection`

The service `com.sun.star.sdb.Connection` has three main functions: communication, data definition and operation on the OpenOffice.org API application level. The service:

- Handles the communication with a database including statement execution, transactions, database metadata and warnings through the simple connection service of the SDBC layer `com.sun.star.sdbc.Connection`.
- Handles database definition tasks, primarily table definitions, through the service `com.sun.star.sdbcx.DatabaseDefinition`. Optionally, it manages views, users and groups.
- Organizes query definitions on the application level and provides a method to open queries and tables defined in OpenOffice.org API. Query definitions are organized by the interfaces `com.sun.star.sdb.XQueriesSupplier` and `com.sun.star.sdb.XSQLQueryComposerFactory`. Queries and tables can be opened using `com.sun.star.sdb.XCommandPreparation`. In case the underlying data source is needed, `com.sun.star.container.XChild` provides the parent data source. This is useful when using an existing connection, for instance, of a database form, to act upon its data source.

Connections are central to all database activities. The connection interfaces are discussed later.

Communication

The main interface of `com.sun.star.sdbc.Connection` is `com.sun.star.sdbc.XConnection`. Its methods control almost every aspect of communication with a database management system:

```
// general connection control
void close()
```

```
boolean isClosed()
void setReadOnly( [in] boolean readOnly)
boolean isReadOnly()

// commands and statements
// - generic SQL statement
// - prepared statement
// - stored procedure call
com::sun::star::sdbc::XStatement createStatement()
com::sun::star::sdbc::XPreparedStatement prepareStatement( [in] string sql)
com::sun::star::sdbc::XPreparedStatement prepareCall( [in] string sql)
string nativeSQL( [in] string sql)

// transactions
void setTransactionIsolation( [in] long level)
long getTransactionIsolation()
void setAutoCommit( [in] boolean autoCommit)
boolean getAutoCommit()
void commit()
void rollback()

// database metadata
com::sun::star::sdbc::XDatabaseMetaData getMetaData()

// data type mapping (driver dependent)
com::sun::star::container::XNameAccess getTypeMap()
void setTypeMap( [in] com::sun::star::container::XNameAccess typeMap)

// catalog (subspace in a database)
void setCatalog( [in] string catalog)
string getCatalog()
```

The use of commands and statements are explained in the sections *12.3 Database Access - Manipulating Data* and *12.4.2 Database Access - Database Design - Using DDL to change the Database Design*. Transactions are discussed in *12.5.1 Database Access - Using DBMS Features - Transaction Handling*. Database metadata are covered in *12.4.1 Database Access - Database Design - Retrieving Information about a Database*.

The `com.sun.star.sdbc.XWarningsSupplier` is a simple interface to handle SQL warnings:

```
any getWarnings()
void clearWarnings()
```

The exception `com.sun.star.sdbc.SQLWarning` is usually not thrown, rather it is transported silently to objects supporting `com.sun.star.sdbc.XWarningsSupplier`. Refer to the API reference for more information about SQL warnings.

Data Definition

The interfaces of `com.sun.star.sdbcx.DatabaseDefinition` are explained in the section *12.4.3 Database Access - Database Design - Using SDBCX to Access the Database Design*.

Operation on Application Level

Handling of query definitions through `com.sun.star.sdb.XQueriesSupplier` and `com.sun.star.sdb.XSQLQueryComposerFactory` is discussed in the section *12.2.2 Database Access - Data Sources in OpenOffice.org API - DataSources - Queries*.

Through `com.sun.star.sdb.XCommandPreparation` get the necessary statement objects to open predefined queries and tables in a data source, and execute arbitrary SQL statements.

```
com::sun::star::sdbc::XPreparedStatement prepareCommand( [in] string command, [in] long commandType)
```

If the value of the parameter `com.sun.star.sdb.CommandType` is `TABLE` or `QUERY`, pass a table name or query name that exists in the `com.sun.star.sdb.DataSource` of the connection. The value `COMMAND` makes `prepareCommand()` expect an SQL string. The result is a prepared statement object that can be parameterized and executed. For details and an example, refer to section *12.3.6 Database Access - Manipulating Data - PreparedStatement From DataSource Queries*.

The interface `com.sun.star.container.XChild` accesses the parent `com.sun.star.sdb.DataSource` of the connection, if available.

```
com::sun::star::uno::XInterface getParent()
void setParent( [in] com::sun::star::uno::XInterface Parent)
```

Connecting Through A DataSource

Data sources in the database context of OpenOffice.org API offer two methods to establish a connection, a non-interactive and an interactive procedure. Use the `com.sun.star.sdbc.XDataSource` interface to connect. It consists of:

```
// establish connection
com::sun::star::sdbc::XConnection getConnection( [in] string user, [in] string password)

// timeout for connection failure
void setLoginTimeout( [in] long seconds)
long getLoginTimeout()
```

If a database does not support logins, pass empty strings to `getConnection()`. For instance, use `getConnection()` against dBase data sources like Bibliography:

```
    XNameAccess xNameAccess = (XNameAccess)UnoRuntime.queryInterface(
        XNameAccess.class, _rMSF.createInstance("com.sun.star.sdb.DatabaseContext"));

    // we use the Bibliography data source
    XDataSource xDS = (XDataSource)UnoRuntime.queryInterface(
        XDataSource.class, xNameAccess.getByName("Bibliography"));

    // simple way to connect
    XConnection xConnection = xDS.getConnection("", "");
```

However if the database expects a login procedure, hard code the user and password, although this is not advisable. Data sources support an advanced login concept. Their interface `com.sun.star.sdb.XCompletedConnection` starts an interactive login, if necessary:

```
com::sun::star::sdbc::XConnection connectWithCompletion(
                        [in] com::sun::star::task::XInteractionHandler handler)
```

When you call `connectWithCompletion()`, OpenOffice.org API shows the common login dialog to the user if the data source property `IsPasswordRequired` is `true`. The login dialog is part of the `com.sun.star.sdb.InteractionHandler` provided by the global service factory.

```
// logs into a database and returns a connection
// expects a reference to the global service manager
com.sun.star.sdbc.XConnection logon(XMultiServiceFactory _rMSF) throws com.sun.star.uno.Exception {

    // retrieve the DatabaseContext and get its com.sun.star.container.XNameAccess interface
    XNameAccess xNameAccess = (XNameAccess)UnoRuntime.queryInterface(
        XNameAccess.class, _rMSF.createInstance("com.sun.star.sdb.DatabaseContext"));

    // get an Adabas D data source Ada01 generated in the GUI
    Object dataSource = xNameAccess.getByName("Ada01");

    // create a com.sun.star.sdb.InteractionHandler and get its XInteractionHandler interface
    Object interactionHandler = _rMSF.createInstance("com.sun.star.sdb.InteractionHandler");
    XInteractionHandler xInteractionHandler = (XInteractionHandler)UnoRuntime.queryInterface(
        XInteractionHandler.class, interactionHandler);

    // query for the XCompletedConnection interface of the data source
    XCompletedConnection xCompletedConnection = (XCompletedConnection)UnoRuntime.queryInterface(
        XCompletedConnection.class, dataSource);

    // connect with interactive login
    XConnection xConnection = xCompletedConnection.connectWithCompletion(xInteractionHandler);

    return XConnection;
}
```

Connecting Using the DriverManager and a Database URL

The database context and establishing connections to a database even if there is no data source for it in OpenOffice.org API can be avoided.

To create a connection ask the driver manager for it. The `com.sun.star.sdbc.DriverManager` manages database drivers. The methods of its interface `com.sun.star.sdbc.XDriverManager` are used to connect to a database using a database URL:

```
// establish connection
com::sun::star::sdbc::XConnection getConnection( [in] string url)
com::sun::star::sdbc::XConnection getConnectionWithInfo( [in] string url,
         [in] sequence < com::sun::star::beans::PropertyValue > info)

// timeout for connection failure
void setLoginTimeout( [in] long seconds)
long getLoginTimeout()
```

Additionally, the driver manager enumerates all available drivers, and is used to register and deregister drivers. A URL that identifies a driver and contains information about the database to connect to must be known. The `DriverManager` chooses the first registered driver that accepts this URL. The following line of code illustrates it generally:

```
Connection xConnection = DriverManager.getConnection(url);
```

The structure of the URL consists of a protocol name, followed by the driver specific sub-protocol. The data source administration dialog shows the latest supported protocols. Some protocols are platform dependent. For example, ADO is only supported on Windows.

The URLs and conditions for the various drivers are explained in section *12.2.3 Database Access - Data Sources in OpenOffice.org API - Connections - Driver Specifics* below.

Frequently a connection needs additional information, such as a user name, password or character set. Use the method `getConnectionWithInfo()` to provide this information. The method `getConnectionWithInfo()` takes a sequence of `com.sun.star.beans.PropertyValue` structs. Usually `user` and `password` are supported. For other connection info properties, refer to the section *12.2.3 Database Access - Data Sources in OpenOffice.org API - Connections - Driver Specifics*. (Database/CodeSamples.java)

```
// create the DriverManager
Object driverManager = xMultiServiceFactory.createInstance("com.sun.star.sdbc.DriverManager");

// query for the interface XDriverManager
com.sun.star.sdbc.XDriverManager xDriverManager;

xDriverManager = (XDriverManager)UnoRuntime.queryInterface(
    XDriverManager.class, driverManager);

if (xDriverManager != null) {
    // first create the database URL
    String adabasURL = "sdbc:adabas::MYDB0";

    // create the necessary sequence of PropertyValue structs for user and password
    com.sun.star.beans.PropertyValue [] adabasProps = new com.sun.star.beans.PropertyValue[] {
        new com.sun.star.beans.PropertyValue("user", 0, "Scott",
            com.sun.star.beans.PropertyState.DIRECT_VALUE),
        new com.sun.star.beans.PropertyValue("password", 0, "huutsch",
            com.sun.star.beans.PropertyState.DIRECT_VALUE)
        };

    // now create a connection to Adabas
    XConnection xConnection = xDriverManager.getConnectionWithInfo(adabasURL, adabasProps);

    if (adabasConnection != null) {
        System.out.println("Connection was created!");

        // now we dispose the connection to close it
        XComponent xComponent = (XComponent)UnoRuntime.queryInterface(
            XComponent.class, xConnection );

        if (xComponent != null) {
            // connection must be disposed to avoid memory leaks
            xComponent.dispose();
            System.out.println("Connection disposed!");
        }
    } else {
        System.out.println("Connection could not be created!");
    }
}
```

Connecting Through a Specific Driver

The second method to create an independent, data-source connection is to use a particular driver implementation, such as writing a driver. There are also several implementations. Create an instance of the driver and ask it for a connection to decide what driver is used: (Database/CodeSamples.java)

```
    // create the Driver using the implementation name
    Object aDriver = xMultiServiceFactory.createInstance("com.sun.star.comp.sdbcx.adabas.ODriver");

    // query for the XDriver interface
    com.sun.star.sdbc.XDriver xDriver;
    xDriver = (XDriver)UnoRuntime.queryInterface(XDriver.class, aDriver);

    if (xDriver != null) {
        // first create the needed url
        String adabasURL = "sdbc:adabas::MYDB0";

        // second create the necessary properties
        com.sun.star.beans.PropertyValue [] adabasProps = new com.sun.star.beans.PropertyValue[] {
            new com.sun.star.beans.PropertyValue("user", 0, "test1",
                com.sun.star.beans.PropertyState.DIRECT_VALUE),
            new com.sun.star.beans.PropertyValue("password", 0, "test1",
                com.sun.star.beans.PropertyState.DIRECT_VALUE)
            };

        // now create a connection to adabas
        XConnection adabasConnection = xDriver.connect(adabasURL,adabasProps);

        if (xConnection != null) {
            System.out.println("Connection was created!");
            // now we dispose the connection to close it
            XComponent xComponent = (XComponent)UnoRuntime.queryInterface(XComponent.class,
xConnection);
            if (xComponent != null) {
                xComponent.dispose();
                System.out.println("Connection disposed!");
            }
        } else {
            System.out.println("Connection could not be created!");
        }
    }
```

Driver Specifics

Currently, there are eight driver implementations. Some support only the simple `com.sun.star.sdbc.Driver` service, some additionally the more extended service from `com.sun.star.sdbcx.Driver` that includes the support for tables, columns, keys, indexes, groups and users. This section describes the capabilities and the missing functionality in some database drivers. Below is a list of all available drivers.

Driver	URL	Solaris	Linux	Windows
JDBC	jdbc:subprotocol:	•	•	•
ODBC 3.5	sdbc:odbc:datasource name	•	•	•
Adabas D	sdbc:adabas:database name	•	•	•
ADO	sdbc:ado:ADO specific			•
dBase	sdbc:dbase:Location of folder or file	•	•	•
Flat file format (csv)	sdbc:flat:Location of folder or file	•	•	•
OpenOffice.org Calc	sdbc:calc:Location of OpenOffice.org Calc file	•	•	•

Driver	URL	Solaris	Linux	Windows
Mozilla addressbook (Mozilla, Outlook, Outlook Express and LDAP)	sdbc:address:Kind of addressbook	•	•	•

The SDBC Driver for JDBC

The SDBC driver for JDBC is a mapping from SDBC API calls to the JDBC API, and vice versa. Basically, this driver is a direct bridge to JDBC. The SDBC driver for JDBC requires a special property called `JavaDriverClass` to know which JDBC driver should be used. The expected value of this property should be the complete class name of the JDBC driver. The following code snippet uses a MySQL JDBC driver to connect.

```
// first create the needed url
String url = "jdbc:mysql://localhost:3306/TestTables";

// second create the necessary properties
com.sun.star.beans.PropertyValue [] props = new com.sun.star.beans.PropertyValue[] {
    new com.sun.star.beans.PropertyValue("user", 0, "test1",
        com.sun.star.beans.PropertyState.DIRECT_VALUE),
    new com.sun.star.beans.PropertyValue("password", 0, "test1",
        com.sun.star.beans.PropertyState.DIRECT_VALUE),
    new com.sun.star.beans.PropertyValue("JavaDriverClass", 0, "org.gjt.mm.mysql.Driver",
        com.sun.star.beans.PropertyState.DIRECT_VALUE)
};

// now create a connection to adabas
xConnection = xDriverManager.getConnectionWithInfo(url, props);
```

Other properties that require setting during the connect process depend on the JDBC driver that is used.

The SDBC Driver for ODBC

This driver is comparable to the SDBC driver for JDBC described above. It maps the ODBC functionality to the SDBC API, but not completely. However, some functionality the SDBC API supports may not work with ODBC, because an ODBC driver may not support this feature and throws an SQL Exception to indicate this. To create a new connection, the driver uses the following URL format:

```
sdbc:odbc: Name of a datasource defined in the system
```

Additionally, this driver supports several properties through the service `com.sun.star.sdbc.ODBCConnectionProperties`. These properties are set while creating a connection:

Properties of `com.sun.star.sdbc.ODBCConnectionProperties`	
`Silent`	`boolean` — If True,the ODBC driver will not be asked for completion. This may happen if the user name and password are already known. Otherwise False.
`Timeout`	`int` — A value corresponding to the number of seconds to wait for any request on the connection to complete before returning to the application.
`UseCatalog`	`boolean` — If `false`,the SDBC driver should not use catalogs. Otherwise True.
`SystemDriverSettings`	`string` — Settings that are submitted to the ODBC driver directly.
`CharSet`	`string` — Converts data from the ODBC driver into the corresponding text encoding. The value must be a value of the list from www.iana.org/assignments/character-sets. Only a few character sets are supported

Properties of com.sun.star.sdbc.ODBCConnectionProperties	
ParameterNameSubstitution	boolean — If True,all occurrences of "?" as a parameter name will be replaced by a valid parameter name. This is for some drivers that mix the order of the parameters.

The SDBC Driver for Adabas D

This driver was the first driver to support the extended service `com.sun.star.sdbcx.Driver`, that offers access to the structure of a database. The Adabas D driver implementation extends the Adabas ODBC driver through knowledge about database structure. The URL should look like this:

```
sdbc:adabas::DATABASENAME
```

or

```
sdbc:adabas:HOST:DATABASENAME
```

To find the correct database name of an Adabas D database in OpenOffice.org API, select **Tools – Data Sources** and look in the **Data source URL** box of the **General** tab. Find the database folders in *sql/wrk* in the Adabas installation folder.

The SDBC Driver for ADO

The SDBC driver for ADO supports the service `com.sun.star.sdbcx.Driver`. ADO does not allow modification on the database structure unless the database is a Jet Engine. Information about the limitations for ADO are available on the Internet. The URL for SDBC driver for ADO looks like this:

```
sdbc:ado:<ADO specific connection string>
```

Possible connection strings are:

- `sdbc:ado:PROVIDER=Microsoft.Jet.OLEDB.4.0;DATA SOURCE=c:\northwind.mdb`
- `sdbc:ado:Provider=msdaora;data source=testdb`

The SDBC Driver for dBase

The dBase driver is one of the basic driver implementations and supports the service `com.sun.star.sdbcx.Driver`. This driver has a number of limitations concerning its abilities to modify the database structure and the extent of its SQL support. The URL for this driver is:

```
sdbc:dbase:<folder or file url>
```

For instance:

```
sdbc:dbase:file:///d:/user/database/biblio
```

Similar to the SDBC driver for ODBC, this driver supports the connection info property `CharSet` to set different text encodings. The second possible property is `ShowDeleted`. When it is set to `true`, deleted rows in a table are still visible. In this state, it is not allowed to delete rows.

The following table shows the shortcomings of the SDBCX part of the dBase driver.

Object	create	alter
table	•	•
column	•	•
key		
index	•	•
group		

Object	create	alter
user		

The driver has the following conditions in its support for SQL statements:

- The `SELECT` statement can not contain more than one table in the `FROM` clause.
- For comparisons the following operators are valid: =, <, >, <>, >=, <=, `LIKE`, `NOT LIKE`, `IS NULL`, `IS NOT NULL`.
- Parameters are allowed, but must be denoted with a leading colon (`SELECT * FROM biblio WHERE Author LIKE :MyParam`) or with a single question mark (`SELECT * FROM biblio WHERE Author LIKE ?`).
- The driver provides a `ResultSet` that supports bookmarks to records.
- The first instance of OpenOffice.org API that accesses a dbase database locks the files for exclusive writing. The lock is never released until the OpenOffice.org API instance, which has obtained the exclusive write access, is closed. This severely limits the access to a dBase database in a network.

The SDBC Driver for Flat File Formats

This driver is another basic driver available in OpenOffice.org API. It can only be used to fetch data from existing text files, and no modifications are allowed, that is, the whole connection is read-only. The URL for this driver is:

```
sdbc:flat:<folder or file url >
```

For instance:

```
sdbc:file:file:///d:/user/database/textbase1
```

Properties that can be set while creating a new connection.

Properties of `com.sun.star.sdbc.FLATConnectionProperties`	
`Extension`	string — Flat file formats are formats such as: • comma separated values format (*.csv) • sdf format (*.sdf) • text file format (*.txt)
`CharSet`	string — Converts data from the ODBC driver into the corresponding text encoding. The value must be a value of the list from www.iana.org/assignments/character-sets. Only some are supported, but a new one can be added.
`FixedLength`	boolean — If true, all occurrences of "?" as a parameter name will be replaced by a valid parameter name. This is necessary, because some drivers mix the order of the parameters.
`HeaderLine`	`boolean` — If `true`, the first line is used for column generation.
`FieldDelimiter`	`string` — Defines a character which should be used to separate fields and columns.
`StringDelimiter`	`string` — Character to identify strings.
`DecimalDelimiter`	`string` — Character to identify decimal values.
`ThousandDelimiter`	`string` — Character to identify the thousand separator. Must be different from `DecimalDelimiter`.

The SDBC Driver for OpenOffice.org Calc Files

This driver is a basic driver for OpenOffice.org Calc files. It can only be used to fetch data from existing tables and no modifications are allowed. The connection is read-only. The URL for this driver is:

```
sdbc:calc:<file url to a OpenOffice.org Calc file or any other extension known by this application>
```

For instance:

```
sdbc:calc:file:///d:/calcfile.sxw
```

The SDBC driver for addressbook

This driver allows OpenOffice.org API to connect to a system addressbook available on the local machine. It supports four different kinds of addressbooks.

Addressbook	Windows	Unix	URL
Mozilla	•	•	sdbc:address:mozilla
LDAP	•	•	sdbc:address:ldap
Outlook Express	•		sdbc:address:outlookexp
Outlook	•		sdbc:address:outlook

All address book variants support read-only access. The driver itself is a wrapper for the Mozilla API.

Connection Pooling

In a basic implementation, there is a 1:1 relationship between the `com.sun.star.sdb.Connection` object used by the client and physical database connection. When the `Connection` object is closed,

the physical connection is dropped, thus the overhead of opening, initializing, and closing the physical connection is incurred for each client session. A *connection pool* solves this problem by maintaining a cache of physical database connections that can be reused across client sessions. Connection pooling improves performance and scalability, particularly in a three-tier environment where multiple clients can share a smaller number of physical database connections. In OpenOffice.org API, the connection pooling is part of a special service called the `ConnectionPool`. This service manages newly created connections and reuses old ones when they are currently unused.

The algorithm used to manage the connection pool is implementation-specific and varies between application servers. The application server provides its clients with an implementation of the `com.sun.star.sdbc.XPooledConnection` interface that makes connection pooling transparent to the client. As a result, the client gets better performance and scalability. When an application is finished using a connection, it closes the logical connection using `close()` at the connection interface `com.sun.star.sdbc.XConnection`. This closes the logical connection, but not the physical connection. Instead, the physical connection is returned to the pool so that it can be reused. Connection pooling is completely transparent to the client: A client obtains a pooled connection from the `com.sun.star.sdbc.ConnectionPool` service calling `getConnectionWithInfo()` at its interface `com.sun.star.sdbc.XDriverManager` and uses it just the same way it obtains and uses a non-pooled connection.

The following sequence of steps outlines what happens when an SDBC client requests a connection from a `ConnectionPool` object:

1. The client obtains an instance of the `com.sun.star.sdbc.ConnectionPool` from the global service manager and calls the same methods on the `ConnectionPool` object as on the `DriverManager`.
2. The application server providing the `ConnectionPool` implementation checks its connection pool for a suitable `PooledConnection` object, a physical database connection, that is available. Determining the suitability of a given `PooledConnection` object includes matching the client's user authentication information or application type, as well as using other implementation-specific criteria. The lookup method and other methods associated with managing the connection pool are specific to the application server.
3. If there are no suitable `PooledConnection` objects available, the application server creates a new physical connection and returns the `PooledConnection`. The `ConnectionPool` is not driver specific. It is implemented in a service called `com.sun.star.sdbc.ConnectionPool`.
4. Regardless if the `PooledConnection` has been retrieved from the pool or created, the application server does internal recording to indicate that the physical connection is now in use.
5. The application server calls the method `PooledConnection.getConnection()` to get a logical `Connection` object. This logical `Connection` object is a *handle* to a physical `PooledConnection` object. This handle is returned by the `XDriverManager` method `getConnectionWithInfo()` when connection pooling is in effect.
6. The logical `Connection` object is returned to the SDBC client that uses the same Connection API as in the standard situation without a `ConnectionPool`. Note that the underlying physical connection cannot be reused until the client calls the `XConnection` method `close()`.

In OpenOffice.org API, connection pooling is enabled by default and can be controlled through **Tools – Options – Data Sources – Connections**. If a connection from a data source defined in OpenOffice.org API is returned, this setting applies to your connection, as well. To take advantage of the pool independently of OpenOffice.org API data sources, use the `com.sun.star.sdbc.ConnectionPool` instead of the `DriverManager`.

Piggyback Connections

Occasionally, there may already be a connected database row set and you want to use its connection. For instance, if a user has opened a database form. To access the same database as the row set of the form, use the connection the form is working with, not opening a second connection. For this purpose, the `com.sun.star.sdb.RowSet` has a property `ActiveConnection` that returns a connection.



Be aware of the fact that the row set owns the connection it uses. That means, once the row set is deleted, the connection is no longer valid.

12.3 Manipulating Data

There are two possibilities to manipulate data in a database with the OpenOffice.org database connectivity.

- Use the `com.sun.star.sdb.RowSet` service that allows using data sources defined in OpenOffice.org through their tables or queries, or through SQL commands.
- Communicate with a database directly using a `Statement` object.

This section describes both possibilities.

12.3.1 The RowSet Service

The service `com.sun.star.sdb.RowSet` is a high-level client side row set that retrieves its data from a database table, a query, an SQL command or a row set reader, which does not have to support SQLl. It is a `com.sun.star.sdb.ResultSet`.

The connection of the row set is a named `DataSource`, the URL of a data access component, or a previously instantiated connection. Depending on the property `ResultSetConcurrency`, the row set caches all data or uses an optimized method to retrieve data, such as refreshing rows by their keys or their bookmarks. In addition, it provides events for row set navigation and row set modifications to approve the actions, and to react upon them.

The row set can be in two different states, before and after execution. Before execution, set all the properties the row set needs for its work. After calling `execute()` on the `RowSet`, move through the result set, or update and delete rows.

Usage

To use a row set, create a `RowSet` instance at the global service manager through the service name `com.sun.star.sdb.RowSet`. Next, the `RowSet` needs a *connection* and a *command* before it can be executed. These have to be configured through `RowSet` properties.

Connection

There are three different ways to establish a connection:

- Setting `DataSourceName` to a data source from the database context. If the `DataSourceName` is not a URL, then the `RowSet` uses the name to get the `DataSource` from the `DatabaseContext` to create a connection to that data source.

- Setting `DataSourceName` to a database URL. The row set tries to use this URL to establish a connection. Database URLs are described in *12.2.3 Database Access - Data Sources in OpenOffice.org API - Connections - Connecting Using the DriverManager and a Database URL*.
- Setting `ActiveConnection` makes a row set ready for immediate use. The row set uses this connection.

The difference between the two properties is that in the first case the `RowSet` owns the connection. The `RowSet` disposes the connection when it is disposed. In the second case, the `RowSet` only *uses* the connection. The user of a `RowSet` is responsible for the disposition of the connection. For a simple `RowSet`, use `DataSourceName`, but when sharing the connection between different row sets, then use `ActiveConnection`.

If there is already a connection, for example, the user opened a database form, open another row set based upon the property `ActiveConnection` of the form. Put the `ActiveConnection` of the form into the `ActiveConnection` property of the new row set.

Command

With a connection and a command, the row set is ready to be executed calling `execute()` on the `com.sun.star.sdbc.XRowSet` interface of the row set. For interactive logon, use `executeWithCompletion()`, see *12.2.3 Database Access - Data Sources in OpenOffice.org API - Connections - Connecting Through a DataSource*. If interactive logon is not feasible for your application, the properties `User` and `Password` can be used to connect to a database that requires logon.

Once the method for how `RowSet` creates it connections has been determined, the properties `Command` and `CommandType` have to be set. The `CommandType` can be TABLE, QUERY or COMMAND where the `Command` can be a table or query name, or an SQL command.

The following table shows the properties supported by `com.sun.star.sdb.RowSet`.

Properties of `com.sun.star.sdb.RowSet`	
`ActiveConnection`	`com.sun.star.sdbc.XConnection`. The active connection is generated by a `DataSource` or by a URL. It could also be set from the outside. If set from outside, the `RowSet` is not responsible for disposition of the connection.
`DataSourceName`	`string` — The name of the `DataSource` to use. This could be a named `DataSource` or the URL of a data access component.
`Command`	`string` — The `Command` is the command that should be executed. The type of command depends on the `com.sun.star.sdb.CommandType`.
`CommandType`	`com.sun.star.sdb.CommandType` Command type: `TABLE`: indicates the command contains a table name that results in a command like "select * from tablename". `QUERY`: indicates the command contains a name of a query component that contains a certain statement. COMMAND: indicates the command is an SQL-Statement.
`ActiveCommand`	`[readonly] string` — he command which is currently used. `com.sun.star.sdb.CommandType`
`IgnoreResult`	`boolean` — Indicates if all results should be discarded.
`Filter`	`string` — Contains a additional filter for a RowSet.
`ApplyFilter`	`boolean` — Indicates if the filter should be applied. The default is false.
`Order`	An additional sort order definition for a `RowSet`.

Properties of com.sun.star.sdb.RowSet	
Privileges	[readonly] long, constants group com.sun.star.sdbcx.Privilege. Indicates the privileges for insert, update, and delete.
IsModified	[readonly] boolean — Indicates if the current row is modified.
IsNew	[readonly] boolean — Indicates if the current row is the InsertRow and can be inserted into the database.
RowCount	[readonly] boolean — Contains the number of rows accessed in a the data source.
IsRowCountFinal	[readonly] boolean — Indicates if all rows of the RowSet have been counted.
UpdateTableName	string — The name of the table that should be updated. This is used for queries that relate to more than one table.
UpdateSchemaName	string — The name of the table schema.
UpdateCatalogName	string — The name of the table catalog.

The com.sun.star.sdb.RowSet includes the service com.sun.star.sdbc.RowSet and its properties. Important settings such as User and Password come from this service:

Properties of com.sun.star.sdbc.RowSet	
DataSourceName	string — Is the name of a named datasource to use.
URL	string — The connection URL. Can be used instead of the DataSourceName.
Command	string — The command that should be executed.
TransactionIsolation	long — Indicates the transaction isolation level that should be used for the connection, according to com.sun.star.sdbc.TransactionIsolation
TypeMap	com::sun::star::container::XNameAccess. The type map that is used for the custom mapping of SQL structured types and distinct types.
EscapeProcessing	boolean — Determines if escape processing is on or off. If escape scanning is on (the default), the driver does the escape substitution before sending the SQL to the database. This is only evaluated if the CommandType is COMMAND.
QueryTimeOut	long — Retrieves the number of seconds the driver waits for a Statement to execute. If the limit is exceeded, a SQLException is thrown. There is no limitation if set to zero.
MaxFieldSize	long — Returns the maximum number of bytes allowed for any column value. This limit is the maximum number of bytes that can be returned for any column value. The limit applies only to DataType::BINARY , DataType::VARBINARY , DataType::LONGVARBINARY , DataType::CHAR , DataType::VARCHAR , and DataType::LONGVARCHAR columns. If the limit is exceeded, the excess data is silently discarded. There is no limitation if set to zero.
MaxRows	long — Retrieves the maximum number of rows that a ResultSet can contain. If the limit is exceeded, the excess rows are silently dropped. There is no limitation if set to zero.
User	string — Determines the user to open the connection for.
Password	string — Determines the user to open the connection for.
ResultSetType	long — Determine the result set type according to com.sun.star.sdbc.ResultSetType

If the command returns results, that is, it selects data, use XRowSet to manipulate the data, because XRowSet is derived from XResultSet. For details on manipulating a com.sun.star.sdb.ResultSet, see *12.3.3 Database Access - Manipulating Data - Result Sets.*

The code fragment below shows how to create a `RowSet`. (Database/RowSet.java)

```
public static void useRowSet(XMultiServiceFactory _rMSF) throws com.sun.star.uno.Exception {
    // first we create our RowSet object
    XRowSet xRowRes = (XRowSet)UnoRuntime.queryInterface(XRowSet.class,
        _rMSF.createInstance("com.sun.star.sdb.RowSet"));
    System.out.println("RowSet created!");

    // set the properties needed to connect to a database
    XPropertySet xProp = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, xRowRes);
    xProp.setPropertyValue("DataSourceName", "Bibliography");
    xProp.setPropertyValue("Command", "biblio");
    xProp.setPropertyValue("CommandType", new Integer(com.sun.star.sdb.CommandType.TABLE));
    xRowRes.execute();
    System.out.println("RowSet executed!");
    XComponent xComp = (XComponent)UnoRuntime.queryInterface(XComponent.class, xRowRes);
    xComp.dispose();
    System.out.println("RowSet destroyed!");
}
```

The value of the read-only `RowSet` properties is only valid after the first call to `execute()` on the `RowSet`. This snippet shows how to read the privileges out of the `RowSet`: (Database/RowSet.java)

```
public static void showRowSetReadOnlyProps(XMultiServiceFactory _rMSF) throws com.sun.star.uno.Exception
{
    // first we create our RowSet object
    XRowSet xRowRes =
        (XRowSet)UnoRuntime.queryInterface(XRowSet.class_rMSF.createInstance(
            "com.sun.star.sdb.RowSet"));
    System.out.println("RowSet created!");

    // set the properties needed to connect to a database
    XPropertySet xProp = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, xRowRes);
    xProp.setPropertyValue("DataSourceName", "Bibliography");
    xProp.setPropertyValue("Command", "biblio");
    xProp.setPropertyValue("CommandType", new Integer(com.sun.star.sdb.CommandType.TABLE));
    xRowRes.execute();
    System.out.println("RowSet executed!");
    Integer aPriv = (Integer)xProp.getPropertyValue("Privileges");
    int nPriv  = aPriv.intValue();

    if ((nPriv & Privilege.SELECT) == Privilege.SELECT) System.out.println("SELECT");
    if ((nPriv & Privilege.INSERT) == Privilege.INSERT) System.out.println("INSERT");
    if ((nPriv & Privilege.UPDATE) == Privilege.UPDATE) System.out.println("UPDATE");
    if ((nPriv & Privilege.DELETE) == Privilege.DELETE) System.out.println("DELETE");

    XComponent xComp = (XComponent)UnoRuntime.queryInterface(XComponent.class, xRowRes);
    xComp.dispose();
    System.out.println("RowSet destroyed!");
}
```

The next example reads the properties `IsRowCountFinal` and `RowCount`. (Database/RowSet.java)

```
public static void showRowSetRowCount(XMultiServiceFactory _rMSF) throws com.sun.star.uno.Exception {
    // first we create our RowSet object
    XRowSet xRowRes = (XRowSet)UnoRuntime.queryInterface(XRowSet.class,
        _rMSF.createInstance("com.sun.star.sdb.RowSet"));
    System.out.println("RowSet created!");

    // set the properties needed to connect to a database
    XPropertySet xProp = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class,xRowRes);
    xProp.setPropertyValue("DataSourceName","Bibliography");
    xProp.setPropertyValue("Command","biblio");
    xProp.setPropertyValue("CommandType",new Integer(com.sun.star.sdb.CommandType.TABLE));
    xRowRes.execute();
    System.out.println("RowSet executed!");

    // now look if the RowCount is already final
    System.out.println("The RowCount is final: " + xProp.getPropertyValue("IsRowCountFinal"));
    XResultSet xRes = (XResultSet)UnoRuntime.queryInterface(XResultSet.class,xRowRes);
    xRes.last();

    System.out.println("The RowCount is final: " + xProp.getPropertyValue("IsRowCountFinal"));
    System.out.println("There are " + xProp.getPropertyValue("RowCount") + " rows!");

    // now destroy the RowSet
    XComponent xComp = (XComponent)UnoRuntime.queryInterface(XComponent.class,xRowRes);
    xComp.dispose();
    System.out.println("RowSet destroyed!");
}
```

Occasionally, it is useful for the user to be notified when the `RowCount` is final. That is accomplished by adding a `com.sun.star.beans.XPropertyChangeListener` for the property `IsRowCountFinal`.

Events and Other Notifications

[TOPIC:com.sun.star.sdb.XRowSetApproveBroadcaster;com.sun.star.sdb.XRowSetApproveListener;com.sun.star.sdbc.XRowSetListener;com.sun.star.lang.EventObject]The `RowSet` supports a number of events and notifications. First, there is the `com.sun.star.sdb.XRowSetApproveBroadcaster` interface of the `RowSet` that allows the user to add or remove objects derived from the interface `com.sun.star.sdb.XRowSetApproveListener`. The interface `com.sun.star.sdb.XRowSetApproveListener` defines the following methods:

Methods of `com.sun.star.sdb.XRowSetApproveListener`	
`approveCursorMove()`	Called before a `RowSet's` cursor is moved.
`approveRowChange()`	Called before a row is inserted, updated, or deleted.
`approveRowSetChange()`	Called before a `RowSet` is changed or before a `RowSet` is re-executed.

All three methods return a `boolean` value that allows the `RowSet` to continue when it is true, otherwise the current action is stopped.

Additionally, the `RowSet` supports `com.sun.star.sdbc.XRowSet` that allows the user to add objects which are notified when the `RowSet` *has* changed. This has to be a `com.sun.star.sdbc.XRowSetListener`. The methods are:

Methods of `com.sun.star.sdbc.XRowSetListener`	
`cursorMoved`	Called when a `RowSet's` cursor has been moved.
`rowChanged`	Called when a row has been inserted, updated, or deleted.
`rowSetChanged`	Called when the entire row set has changed, or when the row set has been re-executed.

When an event occurs, the appropriate listener method is called to notify the registered listener(s). If a listener is not interested in a particular kind of event, it implements the method for that event as no-op. All listener methods take a `com.sun.star.lang.EventObject` struct that ontains the `RowSet` object which is the source of the event.

The following table lists the order of events after a specific method call on the RowSet. First the movements.

Method Call	Event Call (before)	Event Call (after)
`beforeFirst()` `first()` `next()` `previous()` `last()` `afterLast()` `absolute()` `relative()` `moveToBookmark()` `moveRelativeToBookmark()`	`approveCursorMove()`	`cursorMoved()`, only when the movement was successful `modified()` event from `com.sun.star.beans.XPropertySet` of property RowCount, only when changed `modified()` event from `com.sun.star.beans.XPropertySet` of property RowCountFinal, only when changed
`updateRow()` `deleteRow()` `insertRow()`	`approveRowChange()`	`rowChanged()`
`execute()`	`approveRowSetChange()`	`rowSetChanged()`

Consider a simple class which implements the two listener interfaces described above. (Database/RowSetEventListener.java)

```
import com.sun.star.sdb.XRowSetApproveListener;
import com.sun.star.sdbc.XRowSetListener;
import com.sun.star.sdb.RowChangeEvent;
import com.sun.star.lang.EventObject;

public class RowSetEventListener implements XRowSetApproveListener,XRowSetListener {
    // XEventListener
    public void disposing(com.sun.star.lang.EventObject event) {
        System.out.println("RowSet will be destroyed!");
    }

    // XRowSetApproveBroadcaster
    public boolean approveCursorMove(EventObject event) {
        System.out.println("Before CursorMove!");
        return true;
    }
    public boolean approveRowChange(RowChangeEvent event) {
        System.out.println("Before row change!");
        return true;
    }
    public boolean approveRowSetChange(EventObject event) {
        System.out.println("Before RowSet change!");
        return true;
    }

    // XRowSetListener
    public void cursorMoved(com.sun.star.lang.EventObject event) {
        System.out.println("Cursor moved!");
    }
    public void rowChanged(com.sun.star.lang.EventObject event) {
        System.out.println("Row changed!");
    }
    public void rowSetChanged(com.sun.star.lang.EventObject event) {
        System.out.println("RowSet changed!");
    }
}
```

The following method uses the listener implementation above. (Database/RowSet.java)

```
public static void showRowSetEvents(XMultiServiceFactory _rMSF) throws com.sun.star.uno.Exception {
    // first we create our RowSet object
    XRowSet xRowRes = (XRowSet)UnoRuntime.queryInterface(
        XRowSet.class, _rMSF.createInstance("com.sun.star.sdb.RowSet"));

    System.out.println("RowSet created!");
    // add our Listener
    System.out.println("Append our Listener!");
    RowSetEventListener pRow = new RowSetEventListener();
    XRowSetApproveBroadcaster xApBroad = (XRowSetApproveBroadcaster)UnoRuntime.queryInterface(
        XRowSetApproveBroadcaster.class, xRowRes);
    xApBroad.addRowSetApproveListener(pRow);
    xRowRes.addRowSetListener(pRow);

    // set the properties needed to connect to a database
    XPropertySet xProp = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class,xRowRes);
```

```
    xProp.setPropertyValue("DataSourceName", "Bibliography");
    xProp.setPropertyValue("Command", "biblio");
    xProp.setPropertyValue("CommandType", new Integer(com.sun.star.sdb.CommandType.TABLE));

    xRowRes.execute();
    System.out.println("RowSet executed!");

    // do some movements to check if we got all notifications
    XResultSet xRes = (XResultSet)UnoRuntime.queryInterface(XResultSet.class, xRowRes);
    System.out.println("beforeFirst");
    xRes.beforeFirst();
    // this should lead to no notifications because
    // we should stand before the first row at the beginning
    System.out.println("We stand before the first row: " + xRes.isBeforeFirst());

    System.out.println("next");
    xRes.next();
    System.out.println("next");
    xRes.next();
    System.out.println("last");
    xRes.last();
    System.out.println("next");
    xRes.next();
    System.out.println("We stand after the last row: " + xRes.isAfterLast());
    System.out.println("first");
    xRes.first();
    System.out.println("previous");
    xRes.previous();
    System.out.println("We stand before the first row: " + xRes.isBeforeFirst());
    System.out.println("afterLast");
    xRes.afterLast();
    System.out.println("We stand after the last row: " + xRes.isAfterLast());

    // now destroy the RowSet
    XComponent xComp = (XComponent)UnoRuntime.queryInterface(XComponent.class, xRowRes);
    xComp.dispose();
    System.out.println("RowSet destroyed!");
}
```

Clones of the RowSet Service

Occasionally, a second or third `RowSet` that operates on the same data as the original `RowSet`, is required. This is useful when the rows should be displayed in a graphical representation. For the graphical part a clone can be used which only moves through the rows and displays the data. When a modification occurs on one specific row, the original `RowSet` can be used to do this task.

The new clone is an object that supports the service `com.sun.star.sdb.ResultSet` if it was created using the interface `com.sun.star.sdb.XResultSetAccess` of the original `RowSet`. It is interoperable with the `RowSet` that created it, for example, bookmarks can be exchanged between both sets. If the original `RowSet` has not been executed before, `null` is returned.

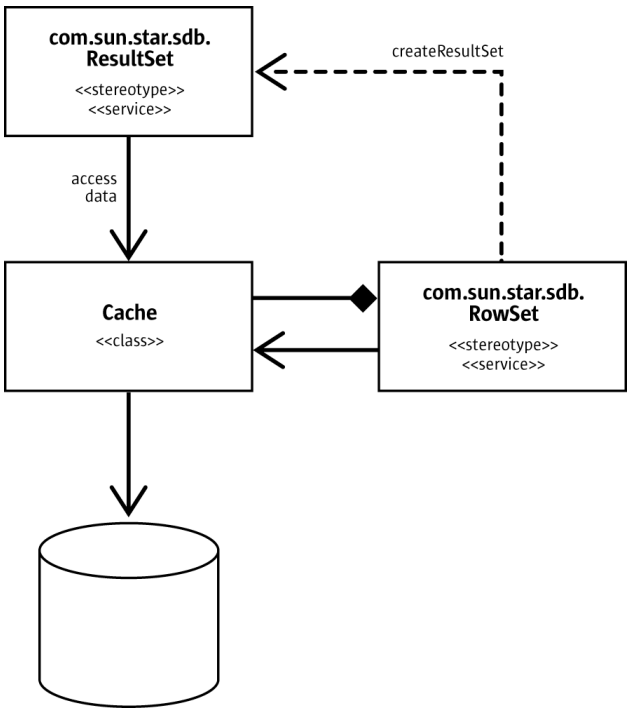


Illustration 177: Data access of RowSet and clone

12.3.2 Statements

The basic procedure to communicate with a database using an SQL statement is always the same:

1. Get a connection object.
2. Ask the connection for a statement.
3. The statement executes a query or an update command. Use the appropriate method to execute the command.
4. If the statement returns a result set, process the result set.

Creating Statements

A `Statement` object is required to send SQL statements to the Database Management System (DBMS). A `Statement` object is created using `createStatement()` at the `com.sun.star.sdbc.XConnection` interface of the connection. It returns a `com.sun.star.sdbc.Statement` service. This `Statement` is generic, that is, it does not contain any SQL command. It can be used for all kinds of SQL commands. Its main interface is `com.sun.star.sdbc.XStatement`:

```
com::sun::star::sdbc::XResultSet executeQuery( [in] string sql)
long executeUpdate( [in] string sql)
boolean execute( [in] string sql)
com::sun::star::sdbc::XConnection getConnection()
```

Once a `Statement` is obtained, choose the appropriate execution method for the SQL command. For a `SELECT` statement, use the method `executeQuery()`. For `UPDATE`, `DELETE` and `INSERT` statements, the proper method is `executeUpdate()`. To have multiple result sets returned, use `execute()` together with the interface `com.sun.star.sdbc.XMultipleResults` of the statement.



Data definition commands, such as `CREATE`, `DROP`, `ALTER`, and `GRANT`, can be issued with `executeUpdate()`.

Consider how an `XConnection` is used to create an `XStatement` in the following example:

```
public static void executeSelect(XMultiServiceFactory _rMSF) throws com.sun.star.uno.Exception {
       // retrieve the DatabaseContext and get its com.sun.star.container.XNameAccess interface
    XNameAccess xNameAccess = (XNameAccess)UnoRuntime.queryInterface(
        XNameAccess.class, _rMSF.createInstance("com.sun.star.sdb.DatabaseContext"));

    // connect
    Object dataSource = xNameAccess.getByName("Ada01");
    XDataSource xDataSource = (XDataSource)UnoRuntime.queryInterface(XDataSource.class, dataSource);
    Object interactionHandler = _rMSF.createInstance("com.sun.star.sdb.InteractionHandler");
    XInteractionHandler xInteractionHandler = (XInteractionHandler)UnoRuntime.queryInterface(
        XInteractionHandler.class, interactionHandler);
    XCompletedConnection xCompletedConnection = (XCompletedConnection)UnoRuntime.queryInterface(
        XCompletedConnection.class, dataSource);
    XConnection xConnection = xCompletedConnection.connectWithCompletion(xInteractionHandler);

    // the connection creates a statement
    XStatement xStatement = xConnection.createStatement();

    // The XStatement interface is used to execute a SELECT command
    // Double quotes for identifiers in the SELECT string must be escaped in Java
    XResultSet xResult = xStatement.executeQuery("Select * from \"Table1\"");

    // process the result ...
    XRow xRow = (XRow)UnoRuntime.queryInterface(XRow.class, xResult);
    while (xResult != null && xResult.next()) {
        System.out.println(xRow.getString(1));
    }
}
```

The remainder of this section discusses how to enter data into a table and retrieving the data later, using `INSERT` and `SELECT` commands with a `com.sun.star.sdbc.Statement`.

Inserting and Updating Data

The following examples use a sample Adabas D database. Generate an Adabas D database in the OpenOffice.org API installation and define a new table named SALESMAN.

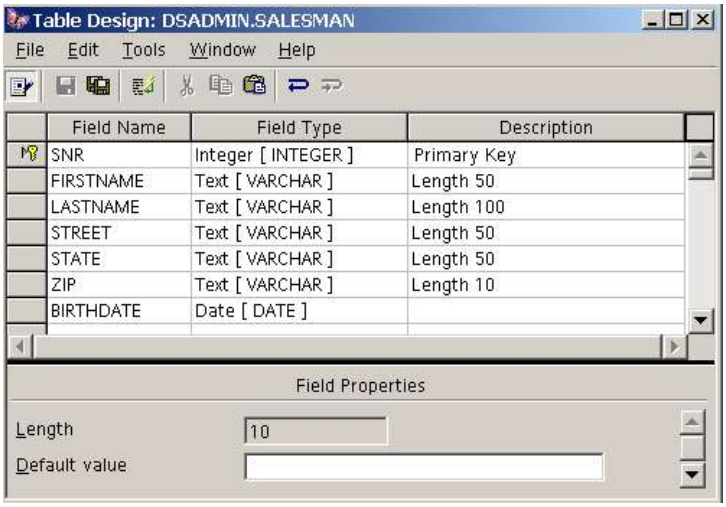


Illustration 178: SALESMAN Table Design

Illustration 175 shows the definition of the SALESMAN table in the OpenOffice.org API data source administrator. The description column shows the lengths defined for the text fields of the table. After all the fields are defined, right-click the row header of the column SNR and choose **Primary Key** to make SNR the primary key. Afterwards a small key icon in the row header shows that SNR is the primary key of the table SALESMAN. When completed, save the table as

SALESMAN. It is important to use uppercase letters for the table name, otherwise the example SQL code will not work.

The table does not contain any data. Use the following `INSERT` command to insert data into the table one row at a time:

```
INSERT INTO SALESMAN (
  SNR,
  FIRSTNAME,
  LASTNAME,
  STREET,
  STATE,
  ZIP,
  BIRTHDATE
  )
VALUES (
  1,
  'Joseph',
  'Smith',
  'Bond Street',
  'CA',
  '95460',
  '1946-07-02'
  )
```



Note the single quotes around the values for the text fields. Single quotes denote character strings in SQL, while double quotes are used for case-sensitive identifiers,such as table and column names.

The following code sample inserts one row of data with the value `1` in the column `SNR`, `'Joseph'` in `FIRSTNAME`, `'Smith'` in `LASTNAME`, with other information in the following columns of the table SALESMAN. To issue the command against the database, create a `Statement` object and then execute it using the method `executeUpdate()`:

```
    XStatement xStatement = xConnection.createStatement();

    xStatement.executeUpdate("INSERT INTO SALESMAN (" +
        "SNR, FIRSTNAME, LASTNAME, STREET, STATE, ZIP, BIRTHDATE) " +
        "VALUES (1, 'Joseph', 'Smith','Bond Street','CA','95460','1946-07-02')");
```

The next call to `executeUpdate()` inserts more rows into the table `SALESMAN`. Note the `Statement` object `stmt` is reused, rather than creating a new one for each update.

```
    xStatement.executeUpdate("INSERT INTO SALESMAN (" +
        "SNR, FIRSTNAME, LASTNAME, STREET, STATE, ZIP, BIRTHDATE) " +
        "VALUES (2, 'Frank', 'Jones', 'Lake Silver', 'CA', '95460', '1963-12-24')");

    xStatement.executeUpdate("INSERT INTO SALESMAN (" +
        "SNR, FIRSTNAME, LASTNAME, STREET, STATE, ZIP, BIRTHDATE) " +
        "VALUES (3, 'Jane', 'Esperanza', '23 Hollywood drive', 'CA', '95460', '1972-01-04')");

    xStatement.executeUpdate("INSERT INTO SALESMAN (" +
        "SNR, FIRSTNAME, LASTNAME, STREET, STATE, ZIP, BIRTHDATE) " +
        "VALUES (4, 'George', 'Flint', '12 Washington street', 'CA', '95460', '1953-02-13')");

    xStatement.executeUpdate("INSERT INTO SALESMAN (" +
        "SNR, FIRSTNAME, LASTNAME, STREET, STATE, ZIP, BIRTHDATE) " +
        "VALUES (5, 'Bob', 'Meyers', '2 Moon way', 'CA', '95460', '1949-09-07')");
```

Updating tables is basically the same process. The SQL command:

```
UPDATE SALESMAN
SET STREET='Grant Street', STATE='FL'
WHERE SNR=2
```

writes a new street and state entry for Frank Jones who has SNR=2. The corresponding `executeUpdate()` call looks like this:

```
    int n = xStatement.executeUpdate("UPDATE SALESMAN " +
        "SET STREET='Grant Street', STATE='FL' " +
        "WHERE SNR=2");
```

The return value of `executeUpdate()` is an `int` that indicates how many rows of a table were updated. Our update command affected one row, so `n` is equal to `1`.



Note that it depends on the database and the driver,if the return value of `executeUpdate()` reflects the actual changes.

Getting Data from a Table

Now that the table `SALESMAN` has values in it, write a `SELECT` statement to access those values. The asterisk * in the following SQL statement indicates that all columns should be selected. Since there is no `WHERE` clause to select less rows, the following SQL statement selects the whole table:

```
SELECT * FROM SALESMAN
```

The result contains the following data:

SNR	FIRSTNAME	LASTNAME	STREET	STATE	ZIP	BIRTHDATE
1	Joseph	Smith	Bond Street	CA	95460	07.02.46
2	Frank	Jones	Lake silver	CA	95460	24.12.63
3	Jane	Esperanza	23 Hollywood drive	CA	95460	01.04.72
4	George	Flint	12 Washington street	CA	95460	13.02.53
5	Bob	Meyers	2 Moon way	CA	95460	07.09.49

The following is another example of a `SELECT` statement. This statement gets a list with the names and addresses of all the salespersons. Only the columns FIRSTNAME, LASTNAME and STREET were selected.

```
SELECT FIRSTNAME, LASTNAME, STREET FROM SALESMAN
```

The result of this query only contains three columns:

FIRSTNAME	LASTNAME	STREET
Joseph	Smith	Bond Street
Frank	Jones	Lake silver
Jane	Esperansa	23 Hollywood drive
George	Flint	12 Washington street
Bob	Meyers	2 Moon way

The `SELECT` statement above extracts all salespersons in the table. The following SQL statement limits the SALESMAN `SELECT` to salespersons who were born before 01/01/1950:

```
SELECT FIRSTNAME, LASTNAME, BIRTHDATE
FROM SALESMAN
WHERE BIRTHDATE < '1950-01-01'
```

The resulting data is:

FIRSTNAME	LASTNAME	BIRTHDATE
Joseph	Smith	02/07/46
Bob	Meyers	09/07/49

When a database is accessed through the OpenOffice.org API database integration, the results are retrieved through `ResultSet` objects. The next section discusses how to use result sets. The following `executeQuery()` call executes the SQL command above. Note that the `Statement` is used again: (Database/Sales.java)

```
com.sun.star.sdbc.XResultSet xResult = xStatement.executeQuery("SELECT FIRSTNAME, LASTNAME, BIRTHDATE "
+
        "FROM SALESMAN " +
        "WHERE BIRTHDATE < '1950-01-01'");
```

12.3.3 Result Sets

The `ResultSet` objects represent the output of an `SQL SELECT` command in data rows and columns to retrieve the data using a row cursor that points to one data row at a time. The following illustration shows the inheritance of `com.sun.star.sdb.ResultSet`. Each layer of the OpenOffice.org API database integration adds capabilities to OpenOffice.org API result sets.

The fundamental `com.sun.star.sdbc.ResultSet` is the most powerful of the three result set services. Basically this result set is sufficient to process SELECT results. It is used to navigate through the resulting rows, and to retrieve and update data rows and the column values in a row.

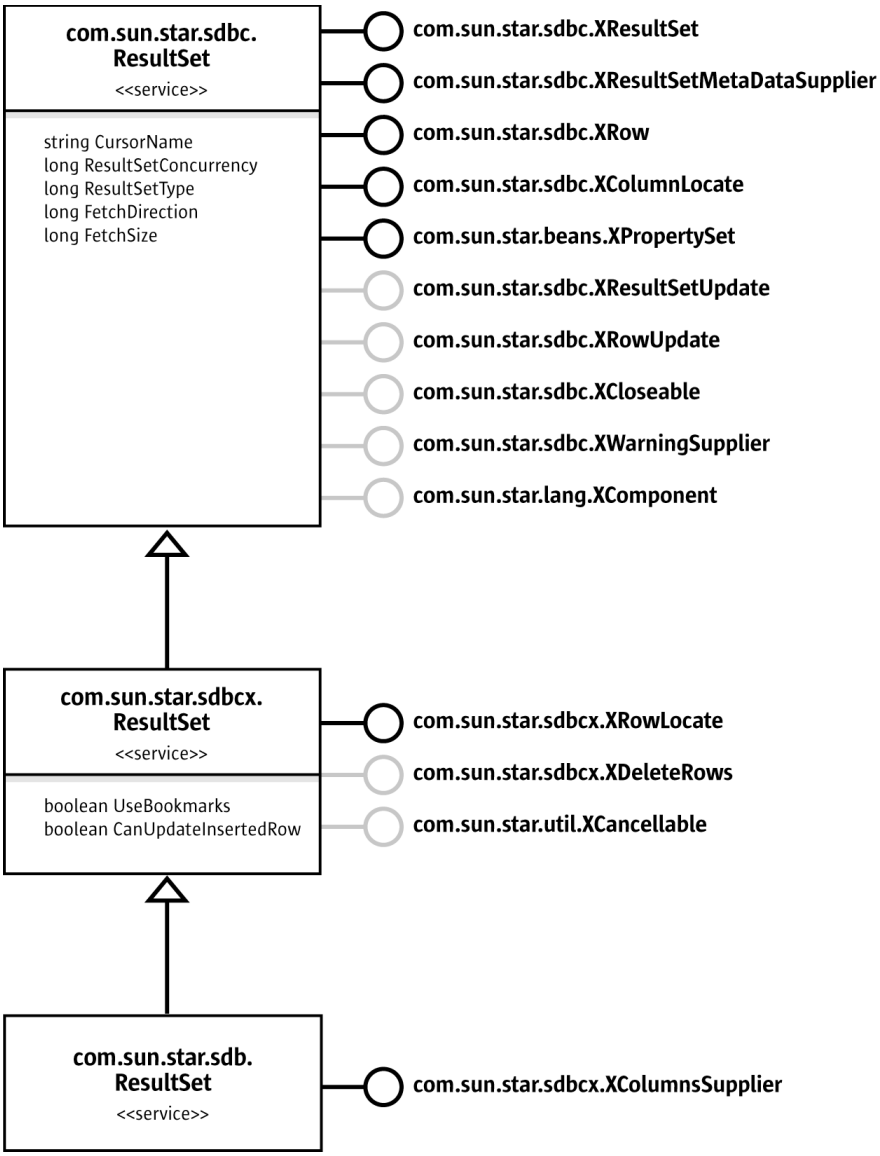


Illustration 179: ResultSet

The `com.sun.star.sdbcx.ResultSet` can add bookmarks through `com.sun.star.sdbcx.XRowLocate` and allows row deletion by bookmarks through `com.sun.star.sdbcx.XDeleteRows`.

The `com.sun.star.sdb.ResultSet` service extends the `com.sun.star.sdbcx.ResultSet` service by the additional interface `com.sun.star.sdbcx.XColumnsSupplier` that allows the user to access information about the appearance of the selected columns in the application. The interface `XColumnsSupplier` returns a `com.sun.star.sdbcx.Container` of `ResultColumns`.

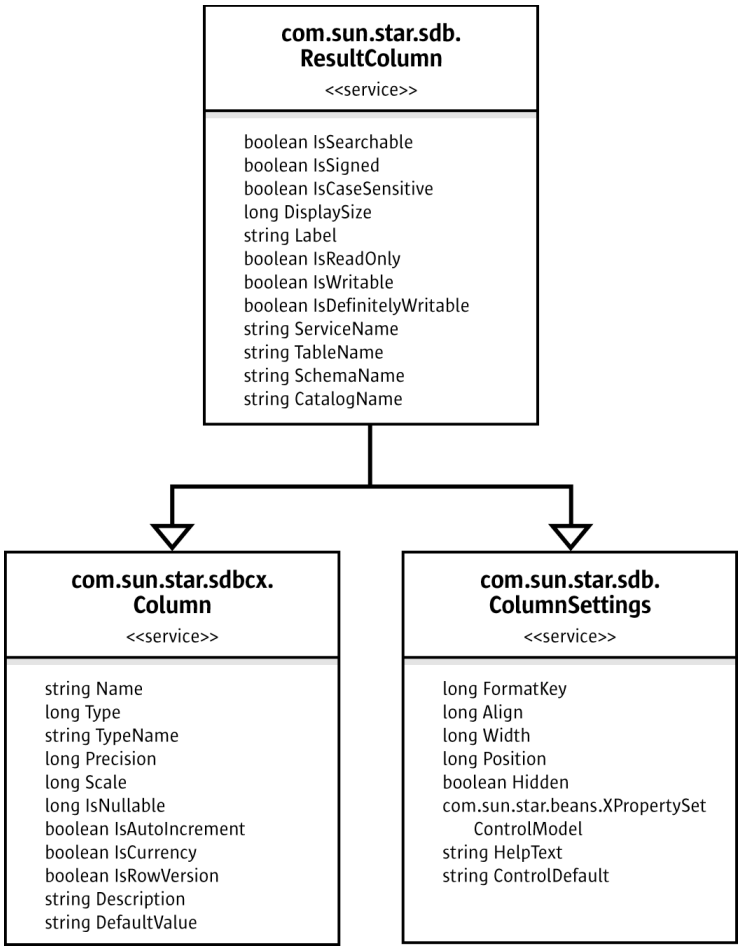


Illustration 180: ResultColumn

The `com.sun.star.sdb.ResultColumn` service inherits the properties of the services `com.sun.star.sdbcx.Column` and `com.sun.star.sdb.ColumnSettings`.

The following table explains the properties introduced with `com.sun.star.sdb.ResultColumn`. For the inherited properties, refer to the section *12.2.2 Database Access - Data Sources in OpenOffice.org API - DataSources - Tables and Columns*.

Properties of `com.sun.star.sdb.ResultColumn`	
`IsSearchable`	boolean —Indicates if the column can be used in a “WHERE” clause.
`IsSigned`	boolean —Indicates if values in the column are signed numbers.
`IsCaseSensitive`	boolean —Indicates if a column is case sensitive.
`DisplaySize`	long —Indicates the column's normal, maximum width in chars.
`Label`	`string` —Gets the suggested column title for use with GUI controls and printouts.
`IsReadOnly`	`boolean` —If `True`, cannot write to the column.
`IsWritable`	`boolean` —If `True`, an attempt to write to the column may succeed.
`IsDefinitelyWritable`	`boolean` —If `True`, the column is writable.
`ServiceName`	`string` —Returns the fully-qualified name of the service that is returned when the `com.sun.star.sdbc.XRow` method `getObject()` is called to retrieve a value from the column.

Properties of com.sun.star.sdb.ResultColumn	
TableName	string—Gets the database table name where the column comes from.
SchemaName	string—Gets the schema name of the column.
CatalogName	string—Gets the catalog name of the column.

Retrieving Values from Result Sets

A call to execute() on a com.sun.star.sdb.RowSet or a call to executeQuery() on a Statement produces a com.sun.star.sdb.ResultSet. (Database/Sales.java)

```
com.sun.star.sdbc.XResultSet xResult = xStatement.executeQuery("SELECT FIRSTNAME, LASTNAME, STREET " +
    "FROM SALESMAN " +
    "VWHERE BIRTHDATE < '1950-01-01'");
```

Moving the Result Set Cursor

The ResultSet stored in the variable xResult contains the following data after the call above:

FIRSTNAME	LASTNAME	BIRTHDATE
Joseph	Smith	02/07/46
Bob	Meyers	09/07/49

To access the data, go to each row and retrieve the values according to their types. The method next() is used to move the row cursor from row to row. Since the cursor is initially positioned just above the first row of a ResultSet object, the first call to next() moves the cursor to the first row and makes it the current row. For the same reason, use the method next() to access the first row even if there is only one row in a result set. Subsequent invocations of next() move the cursor down one row at a time.

The interface com.sun.star.sdbc.XResultSet offers methods to move to specific row numbers, and to positions relative to the current row, in addition to moving the cursor back and forth one row at a time:

```
// move one row at a time
boolean next()
boolean previous()

// move a number of rows
boolean absolute( [in] long row )
boolean relative( [in] long rows )

// move to result set borders and beyond
boolean first()
boolean last()
void beforeFirst()
void afterLast()

//detect position
boolean isBeforeFirst()
boolean isAfterLast()
boolean isFirst()
boolean isLast()
long getRow()

// refetch row from the database
void refreshRow()

// row has been updated, inserted or deleted
boolean rowUpdated()
boolean rowInserted()
boolean rowDeleted()

// get the statement which created the result set
```

```
com::sun::star::uno::XInterface getStatement()
```



Note that you can only move the cursor backwards if you set the statement property `ResultSetType` to `SCROLL_INSENSITIVE` or `SCROLL_SENSITIVE`. For details, refer to chapter *12.3.3 Database Access - Manipulating Data - Result Sets - Scrollable Result Sets*.

Using the getXXX Methods

To get column values from the current row, use the interface `com.sun.star.sdbc.XRow`. It offers a large number of get methods for all SDBC data types, or rather getXXX methods. The XXX stands for the type retrieved by the method.

Usually, the `getXXX` method is used for the appropriate type to retrieve the value in each column. For example, the first column in each row of `xResult` is FIRSTNAME. It is the first column and contains a value of SQL type `VARCHAR`. The appropriate method to retrieve a VARCHAR value is `getString()`. It should be used for the second column, as well. The third column BIRTHDATE stores DATE values, the method for date types is `getDate()`. SDBC is flexible and allows a number of type conversions through getXXX. See the table below for details.

The following code accesses the values stored in the current row of `xResult` and prints a line with the column values separated by tabs. Each time `next()` is invoked, the next row becomes the current row, and the loop continues until there are no more rows in `xResult`.
(Database/SalesMan.java)

```
public static void selectSalespersons(XMultiServiceFactory _rMSF) throws com.sun.star.uno.Exception {
    // retrieve the DatabaseContext and get its com.sun.star.container.XNameAccess interface
    XNameAccess xNameAccess = (XNameAccess)UnoRuntime.queryInterface(
        XNameAccess.class, _rMSF.createInstance("com.sun.star.sdb.DatabaseContext"));

    //connect
    Object dataSource = xNameAccess.getByName("Ada01");
    XDataSource xDataSource = (XDataSource)UnoRuntime.queryInterface(XDataSource.class, dataSource);
    Object interactionHandler = _rMSF.createInstance("com.sun.star.sdb.InteractionHandler");
    XInteractionHandler xInteractionHandler = (XInteractionHandler)UnoRuntime.queryInterface(
        XInteractionHandler.class, interactionHandler);
    XCompletedConnection xCompletedConnection = (XCompletedConnection)UnoRuntime.queryInterface(
        XCompletedConnection.class, dataSource);
    XConnection xConnection = xCompletedConnection.connectWithCompletion(xInteractionHandler);

    // create statement and execute query
    XStatement xStatement = xConnection.createStatement();
    XResultSet xResult = xStatement.executeQuery("SELECT FIRSTNAME, LASTNAME, BIRTHDATE FROM SALESMAN");

    // process result
    XRow xRow = (XRow)UnoRuntime.queryInterface(XRow.class, xResult);
    while (xResult != null && xResult.next()) {
        String firstName = xRow.getString(1);
        String lastName = xRow.getString(2);
        com.sun.star.util.Date birthDate = xRow.getDate(3);
        System.out.println(firstName + "\t" + lastName + "\t\t" +
            birthDate.Month + "/" + birthDate.Day + "/" + birthDate.Year);
    }
}
```

The output looks like this:

```
Joseph  Smith           7/2/1946
Frank   Jones           12/24/1963
Jane    Esperanza       4/1/1972
George  Flint           2/13/1953
Bob     Meyers          9/7/1949
```

In this code, how the `getXXX` methods work are shown and the two `getXXX` calls are examined.

```
String firstName = xRow.getString(1);
```

The method `getString()` is invoked on `xRow`, that is, `getString()` gets the value stored in column no. 1 in the current row of `xResult`, which is FIRSTNAME. The value retrieved by `getString()` has been converted from a `VARCHAR` to a `String` in the Java programming language, and assigned to the `String` object `firstname`.

The situation is similar with the method `getDate()`. It retrieves the value stored in column no. 3 (BIRTHDATE), which is an SQL `DATE`, and converts it to a `com.sun.star.util.Date` before assigning it to the variable `birthDate`.

Note that the column number refers to the column number in the result set, not in the original table.

SDBC is flexible as to which `getXXX` methods can be used to retrieve the various SQL types. For example, the method `getInt()` can be used to retrieve any of the numeric or character types. The data it retrieves is converted to an `int`; that is, if the SQL type is `VARCHAR`, SDBC attempts to parse an integer out of the `VARCHAR`. To be sure that no information is lost, the method `getInt()` is only recommended for SQL `INTEGER` types, and it cannot be used for the SQL types `BINARY`, `VARBINARY`, `LONGVARBINARY`, `DATE`, `TIME`, or `TIMESTAMP`.

Although `getString()` is recommended for the SQL types `CHAR` and `VARCHAR`, it is possible to retrieve any of the basic SQL types with it. The new SQL3 data types can not be retrieved with it. Getting values with `getString()` can be useful, but has its limitations. For instance, if it is used to retrieve a numeric type, `getString()` converts the numeric value to a Java `String` object, and the value has to be converted back to a numeric type before it can be used for numeric operations.

The value will be treated as a string, so if an application is to retrieve and display arbitrary column values of any standard SQL type other than SQL3 types, use `getString()`.

shows all getXXX() methods and the corresponding SDBC data types defined in `com.sun.star.sdbc.DataType`. The illustration above shows which methods can legally be used to retrieve SQL types, and which methods are recommended for retrieving the various SQL types.

	TINYINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC	BIT	CHAR	VARCHAR	LONGVARCHAR	BINARY	VARBINARY	LONGVARBINARY	DATE	TIME	TIMESTAMP
getByte ()	X	X	X	X	X	X	X	X	X	X	X	X	X						
getShort ()	X	X	X	X	X	X	X	X	X	X	X	X	X						
getInt ()	X	X	X	X	X	X	X	X	X	X	X	X	X						
getLong ()	X	X	X	X	X	X	X	X	X	X	X	X	X						
getFloat ()	X	X	X	X	X	X	X	X	X	X	X	X	X						
getDouble ()	X	X	X	X	X	X	X	X	X	X	X	X	X						
getBoolean ()	X	X	X	X	X	X	X	X	X	X	X	X	X						
getString ()	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
getBytes ()														X	X	X			
getDate ()										X	X	X					X		X
getTime ()										X	X	X						X	X
getTimestamp ()										X	X	X					X	X	X
getCharacterStream ()										X	X	X	X	X	X				
getUnicodeStream ()										X	X	X	X	X	X				
getBinaryStream ()													X	X	X				
getObject ()	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Illustration 181: Methods to Retrieve SQL Types

- x with grey background indicates that the getXXX() method is the recommended method to retrieve an SDBC data type. No data will be lost due to type conversion.
- x indicates that the getXXX() method may legally be used to retrieve the given SDBC type. However, type conversion will take place and affect the values you obtain.

Scrollable Result Sets

The interface `com.sun.star.sdbc.XResultSet` offers methods to move the cursor back and forth to an arbitrary row, and get the current position of the cursor. Scrollable result sets are necessary to create GUI tools that can browse result sets. It also may be required to move a specific row to work with it. Before taking advantage of these features, create a scrollable `ResultSet` object. The following lines of code illustrate one way to create a scrollable `ResultSet` object:

```
    XStatement xStatement = xConnection.createStatement();
    XPropertySet xProp = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, xStatement);

    xProp.setPropertyValue("ResultSetType",new java.lang.Integer(ResultSetType.SCROLL_INSENSITIVE));
    xProp.setPropertyValue("ResultSetConcurrency", new java.lang.Integer
(ResultSetConcurrency.UPDATABLE));

    XResultSet xResult = xStatement.executeQuery("SELECT FIRSTNAME, LASTNAME FROM SALES");
```

This code is similar to what was used earlier, except that it sets two property values at the `Statement` object. These properties have to be set before the statement is executed.

The value of the property ResultSetType must be one of three constants defined in `com.sun.star.sdbc.ResultSetType`: `FORWARD_ONLY`, `SCROLL_INSENSITIVE` and `SCROLL_SENSITIVE`.

The property `ResultSetConcurrency` must be one out of the two `com.sun.star.sdbc.ResultSetConcurrency` constants `READ_ONLY` and `UPDATABLE`. When a `ResultSetType` is specified, it must be specified if it is read-only or modifiable.

If any constants for the type and modifiability of a `ResultSet` object are not specified, `FORWARD_ONLY` and `READ_ONLY` will automatically be created.

Specifying the constant `FORWARD_ONLY` creates a non-scrollable result set, that is, the cursor moves forward only. A scrollable `ResultSet` is obtained by specifying `SCROLL_INSENSITIVE` or `SCROLL_SENSITIVE`. Sensitive or insensitive refers to changes made to the underlying data after the result set has been opened. A `SCROLL_INSENSITIVE` result set does not reflect changes to the underlying data, while a `SCROLL_SENSITIVE` result set shows changes. However, not all drivers and databases support change sensitivity.

In scrollable result sets, the counterpart to `next()` is the method `previous()`, which moves the cursor backward. Both methods return `false` when the cursor goes to the position after the last row or before the first row. This allows them to be used in a `while` loop.

The following two examples show the usage of `next()` and `previous()` together with `while`: (Database/Sales.java)

```
    XStatement stmt = con.createStatement();

    XPropertySet xProp = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, stmt);
    xProp.setPropertyValue("ResultSetType",new java.lang.Integer(ResultSetType.SCROLL_INSENSITIVE));
    xProp.setPropertyValue("ResultSetConcurrency", new java.lang.Integer
(ResultSetConcurrency.READ_ONLY));

    XResultSet srs = stmt.executeQuery("SELECT NAME, PRICE FROM SALES");

    XRow row = (XRow)UnoRuntime.queryInterface(XRow.class, srs);

    while (srs.next()) {
        String name = row.getString(1);
        float price = row.getFloat(2);
        System.out.println(name + "     " + price);
}
```

The printout will look similar to this:

```
Linux   32
Beef    15.78
Orange juice    1.50
```

To process the rows going backward, the cursor must start out after the last row. The cursor is moved to the position after the last row with the method `afterLast()`. Then `previous()` moves the cursor from the position after the last row to the last row, and then up to the first row with each iteration through the `while` loop. The loop ends when the cursor reaches the position before the first row, where `previous()` returns `false`. (Database/Sales.java)

```
    XStatement stmt = con.createStatement();

    XPropertySet xProp = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class,stmt);
    xProp.setPropertyValue("ResultSetType", new java.lang.Integer(ResultSetType.SCROLL_INSENSITIVE));
    xProp.setPropertyValue("ResultSetConcurrency", new java.lang.Integer
(ResultSetConcurrency.READ_ONLY));

    XResultSet srs = stmt.executeQuery("SELECT NAME, PRICE FROM SALES");

    XRow row = (XRow)UnoRuntime.queryInterface(XRow.class, srs);

    srs.afterLast();
    while (srs.previous()) {
        String name = row.getString(1);
        float price = row.getFloat(2);
        System.out.println(name + "     " + price);
}
```

The printout will look similar to this:

```
Orange juice    1.50
Beef    15.78
Linux   32
```

The column values are the same, but the rows are in the reverse order.

The cursor can be moved to a specific row in a `ResultSet` object. The methods `first()`, `last()`, `beforeFirst()`, and `afterLast()` move the cursor to the row indicated by the method names.

The method `absolute()` moves the cursor to the row number indicated in the argument passed. If the number is positive, the cursor moves the given number from the beginning. Calling `absolute(1)` moves the cursor to the first row. If the number is negative, the cursor moves the given number of rows from the end. Calling `absolute(-1)` sets the cursor to the last row. The following line of code moves the cursor to the fourth row of `srs`:

```
srs.absolute(4);
```

If `srs` has 500 rows, the following line of code moves the cursor to row 497:

```
srs.absolute(-4);
```

The method `relative()` moves the cursor by an arbitrary number of rows from the current row. A positive number moves the cursor forward, and a negative number moves the cursor backwards. For example, in the following code fragment, the cursor moves to the fourth row, then to the first row, and finally to the third row:

```
    srs.absolute(4); // cursor is on the fourth row
    ...
    srs.relative(-3); // cursor is on the first row
    ...
    srs.relative(2); // cursor is on the third row
```

The method `getRow()` returns the number of the current row. For example, use `getRow()` to verify the current position of the cursor in the previous example using the following code:

```
    srs.absolute(4);
    int rowNum = srs.getRow(); // rowNum should be 4
    srs.relative(-3);
    rowNum = srs.getRow(); // rowNum should be 1
    srs.relative(2);
    rowNum = srs.getRow(); // rowNum should be 3
```

Note that some drivers do not support the `getRow` method. They always return 0.

There are four methods to verify if the cursor is at a particular position. The position is stated in their names: `isFirst()`, `isLast()`, `isBeforeFirst()`, and `isAfterLast()`. These methods return a `boolean` that can be used in a conditional statement. For example, the following code fragment

tests if the cursor is after the last row before invoking the method `previous()` in a `while` loop. If the method `isAfterLast()` returns `false`, the cursor is not after the last row, so the method `afterLast` can be invoked. This guarantees that the cursor is after the last row and that using the method `previous()` in the `while` loop stop at every row in `srs`.

```
    if (srs.isAfterLast() == false) {
        srs.afterLast();
    }
    while (srs.previous()) {
        String name = row.getString(1);
        float price = row.getFloat(2);
        System.out.println(name + "     " + price);
    }
```

How to use the two methods from the `XResultSetUpdate` interface to move the cursor: `moveToInsertRow()` and `moveToCurrentRow()` are discussed in the next section. There are examples illustrating why moving the cursor to certain positions may be required.

Modifiable Result Sets

Another feature of SDBC is the ability to update rows in a result set using methods in the programming language, rather than sending an SQL command. Before doing this, a modifiable result set must be created. To create a modifiable result set, supply the `ResultSetConcurrency` constant `UPDATABLE` to the `Statement` property `ResultSetConcurrency`, so that the `Statement` object creates an modifiable `ResultSet` object each time it executes a query.

The following code fragment creates a modifiable `XResultSet` object `rs` . Note that the code also makes `rs` scrollable. A modifiable `ResultSet` object does not have to be scrollable, but when changes are made to a result set, the user may want to move around in it. With a scrollable result set, there is the ability to move to particular rows that you can work with. If the type is `SCROLL_SENSITIVE`, the new value in a row can be obtained after it has changed without refreshing the whole result set.

```
    XStatement stmt = con.createStatement();

    XPropertySet xProp = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, stmt);
    xProp.setPropertyValue("ResultSetType", new java.lang.Integer(ResultSetType.SCROLL_INSENSITIVE));
    xProp.setPropertyValue("ResultSetConcurrency", new java.lang.Integer
(ResultSetConcurrency.UPDATABLE));

    XResultSet rs = stmt.executeQuery("SELECT NAME, PRICE FROM SALES");

    XRow row = (XRow)UnoRuntime.queryInterface(XRow.class, rs);
```

The `ResultSet` object `rs` may look similar to this:

NAME	PRICE
Linux	$30.00
Beef	$15.78
Orange juice	$1.50

The methods can now be used in the `com.sun.star.sdbc.XRowUpdate` interface of the result set to insert a new row into `rs`, delete an existing row from `rs`, or modify a column value in `rs`.

Update

An update is the modification of a column value in the current row. Suppose the price of orange juice is lowered to 0.99. Using the example above, the update would look like this:

```
    stmt.executeUpdate("UPDATE SALES SET PRICE = 0.99" +
        "WHERE SALENR = 4");
```

The following code fragment shows another way to accomplish the same update, this time using SDBC:

```
rs.last();
XRowUpdate updateRow = (XRowUpdate)UnoRuntime.queryInterface(XRowUpdate.class, rs);
updateRow.updateFloat(2, (float)0.99);
```

Update operations in the SDBC API affect column values in the row where the cursor is positioned. In the first line, the `ResultSet` `rs` calls `last()` to move the cursor to the last row where the column `NAME` has the value `Orange juice`. Once the cursor is on the last row, all of the update methods that are called operate on that row until the cursor is moved to another row.

The second line changes the value of the `PRICE` column to 0.99 by calling `updateFloat()`. This method is used because the column value we want to update is a `float` in Java programming language.

The `updateXXX()` methods in `com.sun.star.sdbc.XRowUpdate` take two parameters: the number of the column to update and the new column value. There are specialized `updateXXX()` methods for each data type, such as `updateString()` and `updateInt()`, just like the `getXXX` methods discussed above.

At this point, the price in `rs` for Orange juice is 0.99, but the price in the table `SALES` in the database is still 1.50. To ensure the update takes effect in the database and not just the result set, the `com.sun.star.sdbc.XResultSetUpdate` method `updateRow()` is called. Here is what the code should look like to update `rs` and `SALES`:

```
rs.last();

XRowUpdate updateRow = (XRowUpdate)UnoRuntime.queryInterface(XRowUpdate.class, rs);
updateRow.updateFloat(2, (float)0.99);
XResultSetUpdate updateRs = (XResultSetUpdate)UnoRuntime.queryInterface(XResultSetUpdate.class, rs);

// update the data in DBMS
updateRs.updateRow();
```

If the cursor is moved to a different row before calling `updateRow()`, the update is lost. The update can be cancelled by calling `cancelRowUpdates()`, for instance, the price should have been 0.79 instead of 0.99. The `cancelRowUpdates()` has to be invoked before invoking `updateRow()`. The `cancelRowUpdates()` does nothing when `updateRow()` has been called. Note that `cancelRowUpdates` cancels all the updates in a row, that is, if there were more than one updateXXX method in the row, they are all cancelled.. The following code fragment cancels the update to the price column to 0.99, and then updates it to 0.79:

```
rs.last();

updateRow.updateFloat(2, (float)0.99);
updateRs.cancelRowUpdates();
updateRow.updateFloat(2, (float)0.79);
updateRs.updateRow();
```

In the above example, only one column value is updated, but an appropriate `updateXXX()` method can be called for any or all of the column values in a single row. Updates and related operations apply to the row where the cursor is positioned. Even if there are many calls to `updateXXX` methods, it takes only one call to the method `updateRow()` to update the database with all changes made in the current row.

To update the price for beef as well, move the cursor to the row containing that product. The row for beef immediately precedes the row for orange juice, so the method `previous()` can be called to position the cursor on the row for `Beef`. The following code fragment changes the price in that row to 10.79 in the result set and underlying table in the database:

```
rs.previous();

updateRow.updateFloat(2, (float)10.79);
updateRs.updateRow();
```

All cursor movements refer to rows in a `ResultSet` object, not to rows in the underlying database. If a query selects five rows from a database table, there are five rows in the result set with the first row being row 1, the second row being row 2, and so on. Row 1 can also be identified as the first row, and in a result set with five rows, row 5 is the last.

The order of the rows in the result set has nothing to do with the physical order of the rows in the underlying table. In fact, the order of the rows in a database table is indeterminate. The DBMS keeps track of which rows were selected, and it makes updates to the proper rows, but they may be located anywhere in the table physically. When a row is inserted, there is no way to know where in the table it was inserted.

Insert

The previous section described how to modify a column value using methods in the SDBC API, rather than SQL commands. With the SDBC API, a new row can also be inserted into a table or an existing row deleted programmatically.

Suppose our salesman Bob sold a new product to one of our customers, FTOP Darjeeling tea, and we need to add the new sale to the database. Using the previous example, write code that passes an SQL insert statement to the DBMS. The following code fragment, in which `stmt` is a `Statement` object, shows this approach: (Database/Sales.java)

```
    stmt.executeUpdate("INSERT INTO SALES " +
        "VALUES (4, 102, 5, 'FTOP Darjeeling tea', '2002-01-02',150)");
```

The same thing can be done, without using any SQL commands, by using `ResultSet` methods in the SDBC API. After a `ResultSet` object is obtained with the results from the table `SALES`, build the new row and then insert it into the result set and the table `SALES` in one step. First, build a new row in the *insert row*, a special row associated with every `ResultSet` object. This row is not part of the result set. It can be considered as a separate buffer in which a new row is composed prior to insertion.

The next step is to move the cursor to the insert row by invoking the method `moveToInsertRow`.(). Then set a value for each column in the row that should not be `null` by calling the appropriate `updateXXX()` method for each value. Note that these are the same `updateXXX()` methods used to change a column value in the previous section.

Finally, call `insertRow()` to insert the row that was populated with values into the result set. This method simultaneously inserts the row into the `ResultSet` object, as well as the database table from where the result set was selected.

The following code fragment creates a scrollable and modifiable `ResultSet` object `rs` that contains all of the rows and columns in the table `SALES`: (Database/Sales.java)

```
    XConnection con = XDriverManager.getConnection("jdbc:mySubprotocol:mySubName");
    XStatement stmt = con.createStatement();

    XPropertySet xProp = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, stmt);
    xProp.setPropertyValue("ResultSetType", new java.lang.Integer(ResultSetType.SCROLL_INSENSITIVE));
    xProp.setPropertyValue("ResultSetConcurrency", new java.lang.Integer
(ResultSetConcurrency.UPDATABLE));

    XResultSet rs = stmt.executeQuery("SELECT * FROM SALES");
    XRow row = (XRow)UnoRuntime.queryInterface(XRow.class, rs);
```

The next code fragment uses the X`ResultSetUpdate` interface of `rs` to insert the row for FTOP Darjeeling tea, shown in the SQL code example. It moves the cursor to the insert row, sets the six column values, and inserts the new row into `rs` and `SALES`: (Database/Sales.java)

```
    XRowUpdate updateRow = (XRowUpdate)UnoRuntime.queryInterface(XRowUpdate.class, rs);
    XResultSetUpdate updateRs = (XResultSetUpdate)UnoRuntime.queryInterface(XResultSetUpdate.class, rs);

    updateRs.moveToInsertRow();
```

```
    updateRow.updateInt(1, 4);
    updateRow.updateInt(2, 102);
    updateRow.updateInt(3, 5);
    updateRow.updateString(4, "FTOP Darjeeling tea");
    updateRow.updateDate(5, new Date((short)1, (short)2, (short)2002));
    updateRow.updateFloat(6, 150);

    updateRs.insertRow();
```

The `updateXXX()` methods behave differently from the way they behaved in the update examples. In those examples, the value set with an u`pdateXXX()` method immediately replaced the column value in the result set, because the cursor was on a row in the result set. When the cursor is on the insert row, the value set with an `updateXXX()` method is immediately set, but it is set in the insert row rather than in the result set itself.

In updates and insertions, calling an updateXXX() method does not affect the underlying database table. The method `updateRow()` must be called to have updates occur in the database. For insertions, the method `insertRow()` inserts the new row into the result set and the database at the same time.

If a value is not supplied for a column that was defined to accept SQL `NULL` values, then the value assigned to that column is `NULL` . If a column does not accept null values, an `SQLException` is returned when an `updateXXX()` method is not called to set a value for it. This is also true if a table column is missing in the `ResultSet` object. In the example above, the query was `SELECT * FROM SALES`, which produced a result set with all the columns of all the rows. To insert one or more rows, the query does not have to select all rows, but it is advisable to select all columns. Additionally if the table has many rows, use a `WHERE` clause to limit the number of rows returned by the `SELECT` statement.

After the method `insertRow()` is called, start building another insert row, or move the cursor back to a result set row. Any of the methods can be executed that move the cursor to a specific row, such as `first()`, `last()`, `beforeFirst()`, `afterLast()`, and `absolute()`. The methods `previous()`, `relative()`, and `moveToCurrentRow()` can also be used. Note that only `moveToCurrentRow()` can be invoked as long as the cursor is on the insert row.

When the method `moveToInsertRow()` is called, the result set records which row the cursor is in, that is by definition the current row. As a consequence, the method `moveToCurrentRow()` can move the cursor from the insert row back to the row that was the current row previously. This also explains why the methods `previous()` and `relative()` can be used, because require movement relative to the current row.

Delete

In the previous sections, how to update a column and insert a new row was explained. This section discusses how to modify the `ResultSet` object by deleting a row. The method `deleteRow()` is called to delete the row where the cursor is placed. For example, to delete the fourth row in the `ResultSet` `rs`, the code look like this: (Database/Sales.java)

```
    rs.absolute(4);

    XResultSetUpdate updateRs = (XResultSetUpdate)UnoRuntime.queryInterface(XResultSetUpdate.class, rs);
    updateRs.deleteRow();
```

The fourth row is removed from `rs` and also from the database.

The only issue about deletions is what the `ResultSet` object does when it deletes a row. With some SDBC drivers, a deleted row is removed and no longer visible in a result set. Other SDBC drivers use a blank row as a placeholder (a "hole") where the deleted row used to be. If there is a blank row in place of the deleted row, the method `absolute()` can be used with the original row positions to move the cursor, because the row numbers in the result set are not changed by the deletion.

Remember that different SDBC drivers handle deletions differently. For example, if an application is meant to run with different databases, the code should not depends on holes in a result set.

Seeing Changes in Result Sets

When data is modified in a `ResultSet` object, the change is always visible immediately. That is, if the same query is re-executed, a new result set is produced based on the data currently in a table. This result set reflects the earlier changes.

If the changes made by you or others are visible while the `ResultSet` object is open, is dependent on the DBMS, the driver, and the type of `ResultSet` object.

With a `SCROLL_SENSITIVE ResultSetType` object, the updates to column values are visible. As well, insertions and deletions are visible, but to ensure this information is returned, use the com.sun.star.sdbc.XDatabaseMetaData methods.

The amount of visibility for changes can be regulated by raising or lowering the transaction isolation level for the connection with the database. For example, the following line of code, where `con` is an active `Connection` object, sets the connection's isolation level to `READ_COMMITTED`:

```
    con.setTransactionIsolation(TransactionIsolation.READ_COMMITTED);
```

With this isolation level, the `ResultSet` object does not show changes before they are committed, but it shows changes that may have other consistency problems. To allow fewer data inconsistencies, raise the transaction isolation level to `REPEATABLE_READ`. Note that the higher the isolation level, the poorer the performance. The database and driver also limited what is actually provided. Many programmers use their database's default transaction isolation level. Consult the DBMS manual for more information about transaction isolation levels.

In a `ResultSet` object that is `SCROLL_INSENSITIVE`, changes are not visible while it is still open. Some programmers only use this type of `ResultSet` object to get a consistent view of the data without seeing changes made by others.

The method `refreshRow()` is used to get the latest values for a row straight from the database. This method is time consuming, especially if the DBMS returns multiple rows `refreshRow()` is called. The method `refreshRow()` can be valuable if it is critical to have the latest data. Even when a result set is sensitive and changes are visible, an application may not always see the latest changes that have been made to a row if the driver retrieves several rows at a time and caches them. Thus, using the method `refreshRow()` ensures that only up-to-date data is visible.

The following code sample illustrates how an application might use the method `refreshRow()` when it is critical to see the latest changes. Note that the result set should be sensitive. If the method `refreshRow()` with a `SCROLL_INSENSITIVE ResultSet` is used, `refreshRow()` does nothing. Getting the latest data for the table `SALES` is not realistic with these methods. A more realistic scenario is when an airline reservation clerk needs to ensure that the seat he is about to reserve is still available. (Database/Sales.java)

```
    XStatement stmt = con.createStatement();

    XPropertySet xProp = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, stmt);
    xProp.setPropertyValue("ResultSetType",new java.lang.Integer(ResultSetType.SCROLL_SENSITIVE));
    xProp.setPropertyValue("ResultSetConcurrency", new java.lang.Integer
(ResultSetConcurrency.READ_ONLY));

    XResultSet rs = stmt.executeQuery("SELECT NAME, PRICE FROM SALES");

    XRow row = (XRow)UnoRuntime.queryInterface(XRow.class, rs);

    rs.absolute(4);

    float price1 = row.getFloat(2);
    // do something ...
    rs.absolute(4);
    rs.refreshRow();
```

```
    float price2 = row.getFloat(2);
    if (price2 != price1) {
        // do something ...
    }
```

12.3.4 ResultSetMetaData

When you develop applications that allow users to create their own SQL statements, for example, through a user interface, information about the result set to be displayed is required. For this reason, the result set supports a method to examine the meta data, that is, information about the columns in the result set. This information could cover items, such as the name of the column, if it is null, if it is an auto increment column, or a currency column. For detailed information, see the interface `com.sun.star.sdbc.XResultSetMetaData`. The following code fragment shows the use of the XResultSetMetaData interface: (Database/Sales.java)

```
    XStatement stmt = con.createStatement();

    XPropertySet xProp = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, stmt);
    xProp.setPropertyValue("ResultSetType",new java.lang.Integer(ResultSetType.SCROLL_INSENSITIVE));
    xProp.setPropertyValue("ResultSetConcurrency", new java.lang.Integer
(ResultSetConcurrency.READ_ONLY));

    XResultSet rs = stmt.executeQuery("SELECT NAME, PRICE FROM SALES");
    XResultSetMetaDataSupplier xRsMetaSup = (XResultSetMetaDataSupplier)UnoRuntime.queryInterface(
    XResultSetMetaDataSupplier.class, rs);
    XResultSetMetaData xRsMetaData =  xRsMetaSup.getMetaData();

    int nColumnCount =  xRsMetaData.getColumnCount();

    for (int i=1 ;i <= nColumnCount; ++i) {
        System.out.println("Name: " + xRsMetaData.getColumnName(i) + " Type: " +
        xRsMetaData.getColumnType(i));
}
```

The printout looks similar to this:

```
Name: NAME Type: 12
Name: PRICE Type: 3
```

Notice that the Type returned is the number for the corresponding SQL data type. In this case, VARCHAR has the value 12 and the type 3 is the SQL data type DECIMAL. The whole list of data types can be found at `com.sun.star.sdbc.DataType`.

Note that the `com.sun.star.sdbc.XResultSetMetaData` can be requested *before* you move to the first row.

12.3.5 Using Prepared Statements

Sometimes it is convenient or efficient to use a `PreparedStatement` object to send SQL statements to the database. This special type of statement includes the more general service `com.sun.star.sdbc.Statement` already discussed.

When to Use a PreparedStatement Object

Using a PreparedStatement object reduces execution time, if executing a `Statement` object many times as in the example above.

The main feature of a `PreparedStatement` object is that it is given an SQL statement when it is created, unlike a `Statement` object. This SQL statement is sent to the DBMS right away where it is compiled. As a result, the `PreparedStatement` object contains not just an SQL statement, but an SQL statement that has been precompiled. This means that when the `PreparedStatement` is

executed, the DBMS can run the PreparedStatement's SQL statement without having to analyze and optimize it again.

The PreparedStatement objects can be used for SQL statements without or without parameters. The advantage of using SQL statements with parameters is that the same statement can be used with different values supplied each time it is executed. This is shown in an example in the following sections.

Creating a PreparedStatement Object

Similar to Statement objects, PreparedStatement objects are created using prepareStatement() on a Connection object. Using our open connection con from the previous examples, code could be written like the following to create a PreparedStatement object that takes two input parameters:

```
XPreparedStatement updateStreet = con.prepareStatement(
    "UPDATE SALESMAN SET STREET = ? WHERE SNR = ?");
```

The variable updateStreet now contains the SQL update statement that has also been sent to the DBMS and precompiled.

Supplying Values for PreparedStatement Parameters

Before executing a PreparedStatement object, values to replace the question mark placeholders or named parameters, such as param1 or param2 have to be supplied. This is accomplished by calling one of the setXXX() methods defined in the interface com.sun.star.sdbc.XParameters of the prepared statement. For instance, to substitute a question mark with a value that is a Java int, call setInt(). If the value is a Java String, call the method setString(). There is a setXXX() method for each type in the Java programming language.

Using the PreparedStatement object updateStreet() from the previous example, the following line of code sets the first question mark placeholder to a Java String with a value of '34 Main Road':

```
XParameters setPara = (XParameters)UnoRuntime.queryInterface(XParameters.class, updateStreet);
setPara.setString(1, "34 Main Road");
```

The example shows that the first argument given to a setXXX() method indicates which question mark placeholder should be set, and the second argument contains the value for the placeholder. The next example sets the second placeholder parameter to the Java int 1:

```
setPara.setInt(2, 1);
```

After these values have been set for its two input parameters, the SQL statement in updateStreet is equivalent to the SQL statement in the String object updateString() used in the previous update example. Therefore, the following two code fragments accomplish the same thing:

Code Fragment 1: (Database/Sales.java)

```
String updateString = "UPDATE SALESMAN SET STREET = '34 Main Road' WHERE SNR = 1";
stmt.executeUpdate(updateString);
```

Code Fragment 2: (Database/Sales.java)

```
XPreparedStatement updateStreet = con.prepareStatement(
    "UPDATE SALESMAN SET STREET = ? WHERE SNR = ? ");
XParameters setPara = (XParameters)UnoRuntime.queryInterface(XParameters.class,updateStreet);
setPara.setString(1, "34 Main Road");
setPara.setInt(2, 1);
updateStreet.executeUpdate();
```

The method `executeUpdate()` was used to execute the `Statement stmt` and the `PreparedStatement updateStreet`. Notice that no argument is supplied to `executeUpdate()` when it is used to execute `updateStreet` . This is true because `updateStreet` already contains the SQL statement to be executed.

Looking at the above examples, a `PreparedStatement` object with parameters was used instead of a statement that involves fewer steps. If a table is going to be updated once or twice, a statement is sufficient, but if the table is going to be updated often, it is efficient to use a `PreparedStatement` object. This is especially true in situation where a `for` loop or `while` loop can be used to set a parameter to a succession of values. This is shown later in this section.

Once a parameter has been set with a value, it retains that value until it is reset to another value or the method `clearParameters()` is called. Using the `PreparedStatement` object `updateStreet`, the following code fragment illustrates reusing a prepared statement after resetting the value of one of its parameters and leaving the other one as is:

```
// set the 1st parameter (the STREET column) to Maryland
setPara.setString(1, "Maryland");

// use the 2nd parameter to select George Flint, his unique identifier SNR is 4
setPara.setInt(2, 4);

// write changes to database
updateStreet.executeUpdate();

// changes STREET column back to Michigan road
// the 2nd parameter for SNR still is 4, only the first parameter is adjusted
updateStreet.executeUpdate();
setPara.setString(1, "Michigan road");

// write changes to database
updateStreet.executeUpdate();
```

12.3.6 PreparedStatement From DataSource Queries

Use the `com.sun.star.sdb.XCommandPreparation` to get the necessary statement objects to open predefined queries and tables in a data source, and to execute arbitrary SQL statements:

```
com::sun::star::sdbc::XPreparedStatement prepareCommand([in] string command, [in] long commandType)
```

If the value of the parameter `com.sun.star.sdb.CommandType` is TABLE or QUERY, pass a table name or query name that exists in the `com.sun.star.sdb.DataSource` of the connection. The value `COMMAND` makes `prepareCommand()` expect an SQL string. The result is a prepared statement object that can be parameterized and executed.

The following fragment opens a predefined query in a database Ada01:

```
// retrieve the DatabaseContext and get its com.sun.star.container.XNameAccess interface
XNameAccess xNameAccess = (XNameAccess)UnoRuntime.queryInterface(
    XNameAccess.class, _rMSF.createInstance("com.sun.star.sdb.DatabaseContext"));

Object dataSource = xNameAccess.getByName("Ada01");
XDataSource xDataSource = (XDataSource)UnoRuntime.queryInterface(XDataSource.class, dataSource);
Object interactionHandler = _rMSF.createInstance("com.sun.star.sdb.InteractionHandler");
XInteractionHandler xInteractionHandler = (XInteractionHandler)UnoRuntime.queryInterface(
    XInteractionHandler.class, interactionHandler);

XCompletedConnection xCompletedConnection = (XCompletedConnection)UnoRuntime.queryInterface(
    XCompletedConnection.class, dataSource);

XConnection xConnection = xCompletedConnection.connectWithCompletion(xInteractionHandler);

XCommandPreparation xCommandPreparation = (XCommandPreparation)UnoRuntime.queryInterface(
    XCommandPreparation.class, xConnection);
XPreparedStatement xPreparedStatement = xCommandPreparation.prepareCommand(
    "Query1", CommandType.QUERY);

XResultSet xResult = xPreparedStatement.executeQuery();
XRow xRow = (XRow)UnoRuntime.queryInterface(XRow.class, xResult);
while (xResult != null && xResult.next()) {
  System.out.println(xRow.getString(1));
```

```
    }
```

12.4 Database Design

12.4.1 Retrieving Information about a Database

The `com.sun.star.sdbc.XDatabaseMetaData` interface is implemented by SDBC drivers to provide information about their underlying database. It is used primarily by application servers and tools to determine how to interact with a given data source. Applications may also use `XDatabaseMetaData` methods to get information about a database. The `com.sun.star.sdbc.XDatabaseMetaData` interface includes over 150 methods, that are categorized according to the types of information they provide:

- General information about the database.
- If the database supports a given feature or capability.
- Database limits.
- What SQL objects the database contains and attributes of those objects.
- Transaction support offered by the data source.

Additionally, the `com.sun.star.sdbc.XDatabaseMetaData` interface uses a resultset with more than 40 possible columns as return values in many `com.sun.star.sdbc.XDatabaseMetaData` methods. This section presents an overview of the `com.sun.star.sdbc.XDatabaseMetaData` interface, and provides examples illustrating the categories of metadata methods. For a comprehensive listing, consult the SDBC API specification.

- Creating the XDatabaseMetaData objects

A `com.sun.star.sdbc.XDatabaseMetaData` object is created using the Connection method `getMetaData()`. Once created, it can be used to dynamically discover information about the underlying data source. The following code example creates a `com.sun.star.sdbc.XDatabaseMetaData` object and uses it to determine the maximum number of characters allowed for a table name.

```
    // xConnection is a Connection object
    XDatabaseMetaData dbmd = xConnection.getMetaData();
    int maxLen = dbmd.getMaxTableNameLength();
```

Retrieving General Information

Some `com.sun.star.sdbc.XDatabaseMetaData` methods are used to dynamically discover general information about a database, as well as details about its implementation. Some of the methods in this category are:

- `getURL()`
- `getUserName()`
- `getDatabaseProductVersion()`, `getDriverMajorVersion()` and `getDriverMinorVersion()`
- `getSchemaTerm()`, `getCatalogTerm()` and `getProcedureTerm()`
- `nullsAreSortedHigh()` and `nullsAreSortedLow()`
- `usesLocalFiles()` and `usesLocalFilePerTable()`

- `getSQLKeywords()`

Determining Feature Support

A large group of `com.sun.star.sdbc.XDatabaseMetaData` methods can be used to determine whether a given feature or set of features is supported by the driver or underlying database. Beyond this, some of the methods describe what level of support is provided. Some of the methods that describe support for individual features are:

- `supportsAlterTableWithDropColumn()`
- `supportsBatchUpdates()`
- `supportsTableCorrelationNames()`
- `supportsPositionedDelete()`
- `supportsFullOuterJoins()`
- `supportsStoredProcedures()`
- `supportsMixedCaseQuotedIdentifiers()`

Methods to describe the level of feature support include:

- `supportsANSI92EntryLevelSQL()`
- `supportsCoreSQLGrammar()`

Database Limits

Another group of methods provides the limits imposed by a given database. Some of the methods in this category are:

- `getMaxRowSize()`
- `getMaxStatementLength()`
- `getMaxTablesInSelect()`
- `getMaxConnections()`
- `getMaxCharLiteralLength()`
- `getMaxColumnsInTable()`

Methods in this group return the limit as an `int`. A return value of zero means there is no limit or the limit is unknown.

SQL Objects and their Attributes

Some methods provide information about the SQL objects that populate a given database. This group also includes methods to determine the attributes of those objects. Methods in this group return `ResultSet` objects in which each row describes a particular object. For example, the method `getUDTs()` returns a `ResultSet` object in which there is a row for each user defined type (UDT) that has been defined in the database. Examples of this category are:

- `getSchemas()` and `getCatalogs()`
- `getTables()`
- `getPrimaryKeys()`
- `getColumns()`

- `getProcedures()` and `getProcedureColumns()`
- `getUDTs()`

For example, to display the structure of a table that consists of columns and keys (primary keys, foreign keys), and also indexes defined on the table, the `com.sun.star.sdbc.XDatabaseMetaData` interface is required: (Database/CodeSamples.java)

```
    XDatabaseMetaData dm = con.getMetaData();
    XResultSet rsTables = dm.getTables(null, "%", "SALES", null);
    XRow rowTB = (XRow)UnoRuntime.queryInterface(XRow.class, rsTables);

    while (rsTables.next()) {
        String catalog = rowTB.getString(1);
        if (rowTB.wasNull())
            catalog = null;

        String schema = rowTB.getString(2);
        if (rowTB.wasNull())
            schema = null;

        String table = rowTB.getString(3);
        String type = rowTB.getString(4);
        System.out.println("Catalog: " + catalog +
            " Schema: " + schema + " Table: " + table + "Type: " + type);
        System.out.println("------------------ Columns ------------------");
        XResultSet rsColumns = dm.getColumns(catalog, schema, table, "%");
        XRow rowCL = (XRow)UnoRuntime.queryInterface(XRow.class, rsColumns);
        while (rsColumns.next()) {
            System.out.println("Column: " + rowCL.getString(4) +
            " Type: " + rowCL.getInt(5) + " TypeName: " + rowCL.getString(6) );
         }
    }
```

Another method often used when creating SQL statements is the method `getIdentifierQuoteString()`. This method is always used when table or column names need to be quoted in the SQL statement. For example:

```
    SELECT "Name", "Price" FROM "Sales"
```

In this case, the identifier quotation is the character ". The combination of `XDatabaseMetaData` methods in the following code fragment may be useful to know if the database supports catalogs and/or schemata. (Database/CodeSamples.java)

```
public static String quoteTableName(XConnection con, String sCatalog, String sSchema,
        String sTable) throws com.sun.star.uno.Exception {
    XDatabaseMetaData dbmd = con.getMetaData();
    String sQuoteString = dbmd.getIdentifierQuoteString();
    String sSeparator = ".";
    String sComposedName = "";
    String sCatalogSep = dbmd.getCatalogSeparator();
    if (0 != sCatalog.length() && dbmd.isCatalogAtStart() && 0 != sCatalogSep.length()) {
        sComposedName += sCatalog;
        sComposedName += dbmd.getCatalogSeparator();
    }
    if (0 != sSchema.length()) {
        sComposedName += sSchema;
        sComposedName += sSeparator;
        sComposedName += sTable;
    } else {
        sComposedName += sTable;
    }
    if (0 != sCatalog.length() && !dbmd.isCatalogAtStart() && 0 != sCatalogSep.length()) {
        sComposedName += dbmd.getCatalogSeparator();
        sComposedName += sCatalog;
    }
    return sComposedName;
}
```

12.4.2 Using DDL to Change the Database Design

To show the usage of statements for data definition purposes, we will show how to create the tables in our example database using CREATE statements. The first table, `SALESMAN`, contains

essential information about the salespersons, including the first name, last name, street address, city, and birth date. The table SALESMAN that is described in more detail later, is shown here:

SNR	FIRSTNAME	LASTNAME	STREET	STATE	ZIP	BIRTH DATE
1	0	0	0	0	95460	02/07/46
2	0	0	0	0	95460	12/24/63
3	0	0	0	0	95460	04/01/72
4	0	0	0	0	95460	02/13/53
5	0	0	0	0	95460	09/07/49

The first column is the column SNR of SQL type INTEGER. This column contains a unique number for each salesperson. Since there is a different SNR for each person, the SNR column can be used to uniquely identify a particular salesman,the is, the primary key. If this were not the case, an additional column that is unique would have to be introduced, such as the social security number. The column for the first name is FIRSTNAME that holds values of the SQL type VARCHAR with a maximum length of 50 characters. The third column, LASTNAME, is also a VARCHAR with a maximum length of 100 characters. The STREET and STATE columns are VARCHARs with 50 characters. The column ZIP uses INTEGER and the column BIRTHDATE uses the type DATE. By using the type DATE instead of VARCHAR, the dates of birth can be compared with the current date.

The second table, CUSTOMER, in our database, contains information about customers:

COS_NR	LASTNAME	STREET	CITY	STATE	ZIP
100	0	0	0	0	95199
101	0	0	0	0	95460
102	0	0	0	0	93966

The first column is the personal number COS_NR of our customer. This column is used to uniquely identify the customers, and declare this column to be the primary key. The types of the other columns are identical to the first table, SALESMAN.

Another table to show joins is required. For this purpose, the table SALES is used. This table contains all sales that our salespersons could enter into an agreement with the customers. This table needs a column SALENR to identify each sale, a column for COS_NR to identify the customer and a column SNR for the sales person who made the sale, and the columns that defines the article sold.

SALENR	COS_NR	SNR	NAME	DATE	PRICE
1	100	1	0	02/12/01	$39.99
2	101	2	0	10/18/01	$15.78
3	102	4	Orange juice	08/09/01	$1.50

To show the relationship between the three tables, consider the diagram below.

The table SALES contains the column COS_NR and the column SNR. These two columns can be used in SELECT statements to get data based on the information in this table, for example, all sales made by the salesperson Jane. The column COS_NR is the primary key in the table CUSTOMER and it uniquely identifies each of the customers. The same is true for the column SNR in the table SALESMAN. In the table SALES, the fields COS_NR and SNR are foreign keys. Note that each COS_NR and SNR number may appear more than once in the SALES table, because a third column SALENR was introduced. This is required for a primary key. An example of how to use primary and foreign keys in a SELECT statement is provided later.

The following CREATE TABLE statement creates the table SALESMAN. The entries within the outer pair of parentheses consist of the name of a column followed by a space and the SQL type to be stored in that column. A comma separates the column entries where each entry consists of a column name and SQL type. The type VARCHAR is created with a maximum length, so it takes a parameter indicating the maximum length. The parameter must be in parentheses following the type. The SQL statement shown here specifies that the name in column FIRSTNAME may be up to 50 characters long:

```
CREATE TABLE SALESMAN
(SNR INTEGER NOT NULL,
 FIRSTNAME VARCHAR(50),
 LASTNAME VARCHAR(100),
 STREET VARCHAR(50),
 STATE VARCHAR(50),
 ZIP VARCHAR(10),
 BIRTHDATE DATE,
 PRIMARY KEY(SNR)
)
```



This code does not end with a DBMS statement terminator that can vary from DBMS to DBMS. For example, Oracle uses a semicolon (;) to indicate the end of a statement, and Sybase uses the word go. The driver you are using automatically supplies the appropriate statement terminator, so that you will not need to include it in your SDBC code.

In the CREATE TABLE statement above, key words are printed in capital letters, and each item is on a separate line. SQL does not require the use of these conventions, it makes the statements easier to read. The standard in SQL is that keywords are not case sensitive, therefore, the following SELECT statement can be written in various ways:

```
SELECT "FirstName", "LastName"
FROM "Employees"
WHERE "LastName" LIKE 'Washington'
```

is equivalent to

```
select "FirstName", LastName" from "Employees" where
"LastName" like 'Washington'
```

Single quotes '...' denote a string literal, double quotes mark case sensitive identifiers in many SQL databases.

Requirements can vary from one DBMS to another for identifier names. For example, some DBMSs require that column and table names must be given exactly as they were created in the CREATE TABLE statement, while others do not. We use uppercase letters for identifiers such as SALESMAN, CUSTOMERS and SALES. Another way would be to ask the XDatabaseMetaData interface if the method storesMixedCaseQuotedIdentifiers() returns true, and to use the string that the method getIdentifierQuoteString() returns.

The data types used in our CREATE TABLE statement are the generic SQL types (also called SDBC types) that are defined in the com.sun.star.sdbc.DataType. DBMSs generally uses these standard types.

To issue the commands above against our database, use the connection con to create a statement and the method executeUpdate() at its interface com.sun.star.sdbc.XStatement. In the following code fragment, executeUpdate() is supplied with the SQL statement from the SALESMAN example above: (Database/SalesMan.java)

```
    XStatement xStatement = con.createStatement();
    int n = xStatement.executeUpdate("CREATE TABLE SALESMAN " +
        "(SNR INTEGER NOT NULL, " +
        "FIRSTNAME VARCHAR(50), " +
        "LASTNAME VARCHAR(100), " +
        "STREET VARCHAR(50), " +
        "STATE VARCHAR(50), " +
        "ZIP INTEGER, " +
        "BIRTHDATE DATE, " +
        "PRIMARY KEY(SNR) " +
        ")");
```

The method `executeUpdate()` is used because the SQL statement contained in `createTable-Salesman` is a DDL (data definition language) statement. Statements that create a table, alter a table, or drop a table are all examples of DDL statements, and are executed using the method `executeUpdate()`.

When the method `executeUpdate()` is used to execute a DDL statement, such as CREATE TABLE, it returns zero. Consequently, in the code fragment above that executes the DDL statement used to create the table `SALESMAN`, `n` is assigned a value of `0`.

12.4.3 Using SDBCX to Access the Database Design

The Extension Layer SDBCX

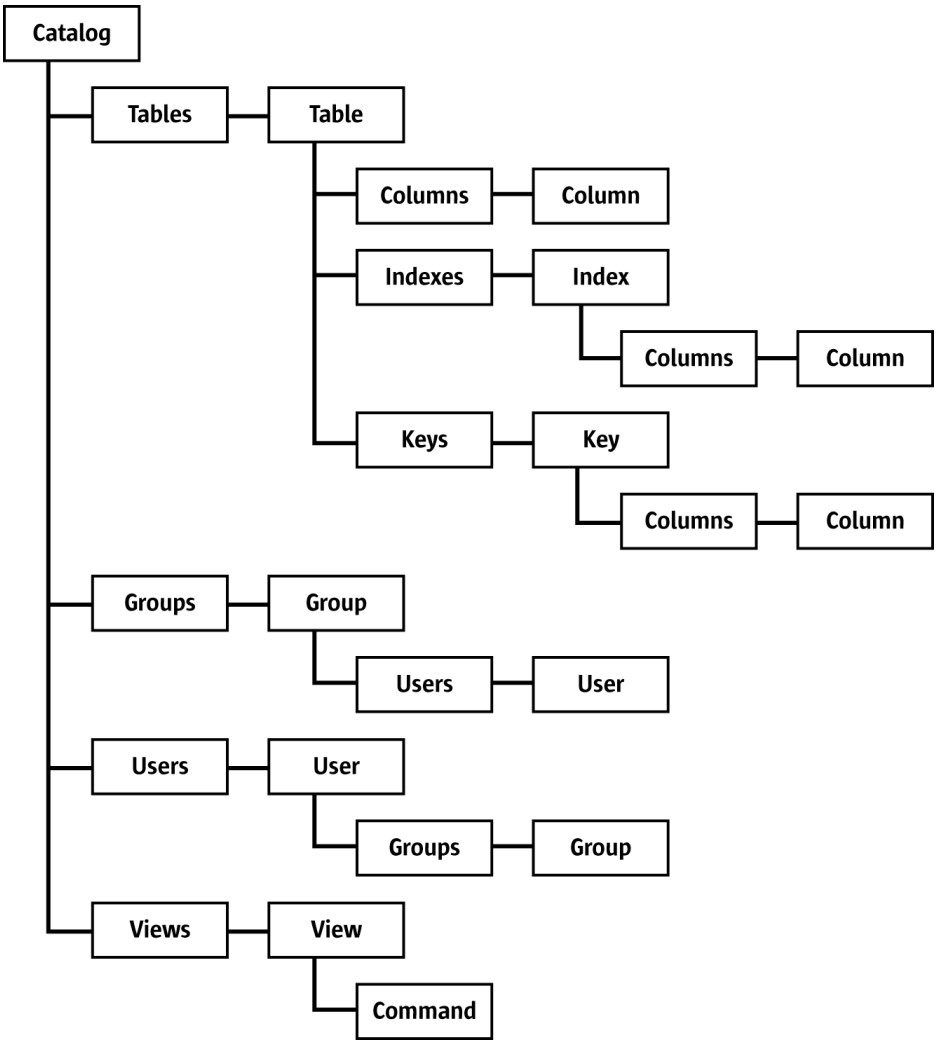


Illustration 182: SDBCX Object design

The SDBCX layer introduces several abstractions built upon the SDBC layer that define general database objects, such as catalog, table, view, group, user, key, index, and column, as well as support for schema and security tasks. These objects are used to manage database design tasks.

The ability of the SDBCX layer to define new data structures makes it an alternative to SQL DDL. The above Illustration 172 gives an overview to the SDBCX objects an their containers.

All objects mentioned previously have matching containers, except for the catalog. Each container implements the service `com.sun.star.sdbcx.Container`. The interfaces that the container supports depend on the objects that reside in it. For instance, the container for keys does not support an `com.sun.star.container.XNameAccess` interface. These containers are used to add and manage new objects in a catalog. The users and groups container manage the control permissions for other SDBCX objects, such as tables and views.

Illustration 171 shows the container specification for SDBCX DatabaseDefinition services.

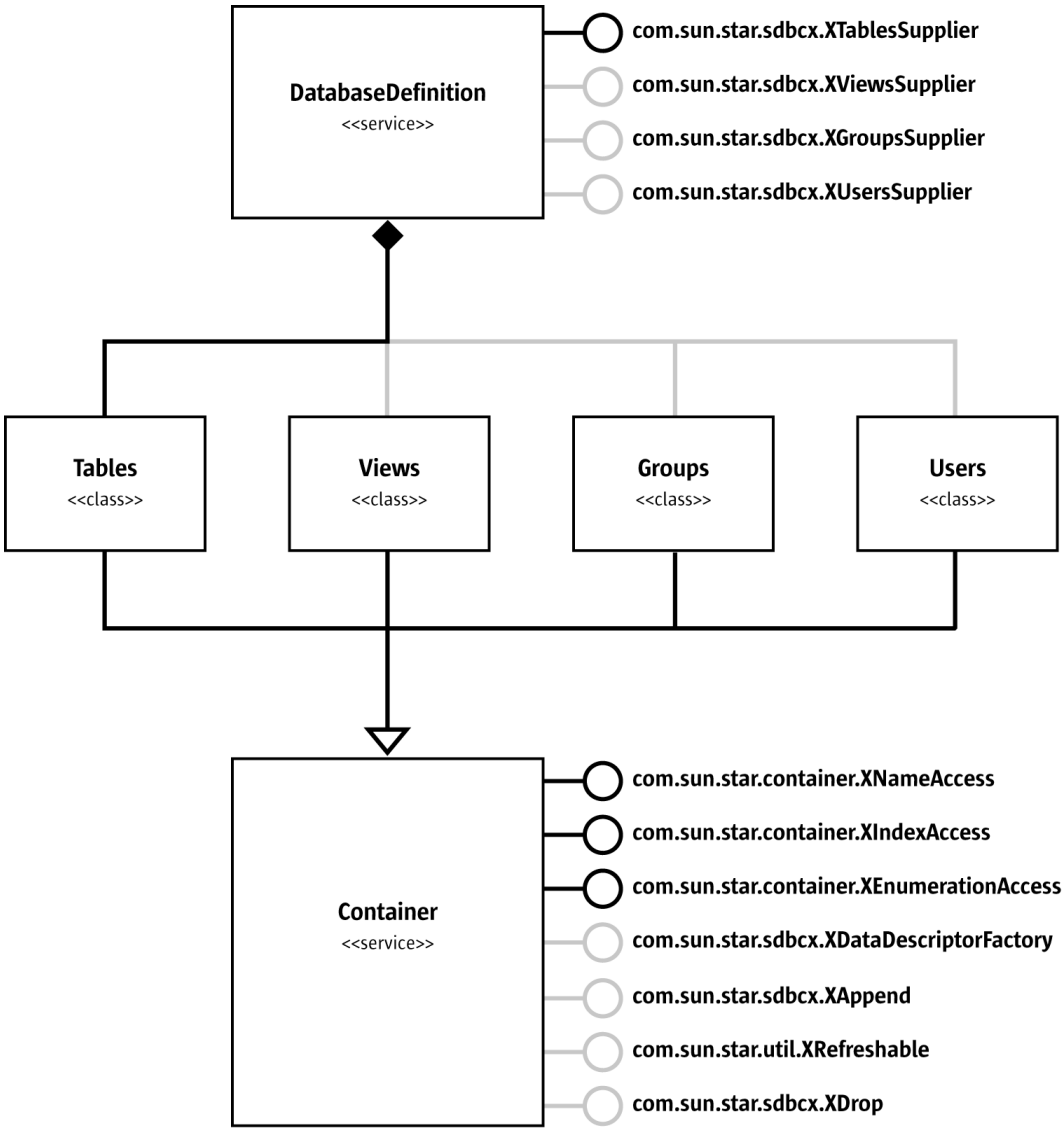


Illustration 183: Database definition

Catalog Service

The Catalog object is the highest-level container in the SDBCX layer. It contains structural features of databases, like the schema and security model for the database. The connection, for instance, represents the database, and the Catalog is the database container for the tables, views, groups,

and users within a connection or database. To create a catalog object, the database driver must support the interface `com.sun.star.sdbcx.XDataDefinitionSupplier` and an existing connection object. The following code fragment lists tables in a database. (Database/sdbcx.java)

```
// create the Driver with the implementation name
Object aDriver = xORB.createInstance("com.sun.star.comp.sdbcx.adabas.ODriver");
// query for the interface
com.sun.star.sdbc.XDriver xDriver;
xDriver = (XDriver)UnoRuntime.queryInterface(XDriver.class, aDriver);
if (xDriver != null) {
    // first create the needed url
    String adabasURL = "sdbc:adabas::MYDB0";
    // second create the necessary properties
    com.sun.star.beans.PropertyValue [] adabasProps = new com.sun.star.beans.PropertyValue[] {
        new com.sun.star.beans.PropertyValue("user", 0, "test1",
            com.sun.star.beans.PropertyState.DIRECT_VALUE),
        new com.sun.star.beans.PropertyValue("password", 0, "test1",
            com.sun.star.beans.PropertyState.DIRECT_VALUE)
    };

    // now create a connection to adabas
    XConnection adabasConnection = xDriver.connect(adabasURL, a dabasProps);
    if(adabasConnection != null) {
        System.out.println("Connection could be created!");
        // we need the XDatabaseDefinitionSupplier interface
        // from the driver to get the XTablesSupplier
        XDataDefinitionSupplier xDDSup = (XDataDefinitionSupplier)UnoRuntime.queryInterface(
            XDataDefinitionSupplier.class, xDriver);
        if (xDDSup != null) {
            XTablesSupplier xTabSup = xDDSup.getDataDefinitionByConnection(adabasConnection);
            if (xTabSup != null) {
                XNameAccess xTables = xTabSup.getTables();
                // now print all table names
                System.out.println("Tables available:");
                String [] aTableNames = xTables.getElementNames();
                for ( int i =0; i<= aTableNames.length-1; i++)
                    System.out.println(aTableNames[i]);
            }
        }
        else {
            System.out.println("The driver is not SDBCX capable!");
        }

        // now we dispose the connection to close it
        XComponent xComponent = (XComponent)UnoRuntime.queryInterface(
            XComponent.class, adabasConnection);
        if (xComponent != null) {
            xComponent.dispose();
            System.out.println("Connection disposed!");
        }
    }
    else {
        System.out.println("Connection could not be created!");
    }
}
```

Table Service

The Table object is a member of the tables container that is a member of the Catalog object. Each Table object supports the same properties, such as `Name`, `CatalogName`, `SchemaName`, `Description`, and an optional `Type`. The properties `CatalogName` and `SchemaName` can be empty when the database does not support these features. The `Description` property contains any comments that were added to the table object at creation time. The optional property `Type` is a string property may contain a database specific table type when supported, . Common table types are "TABLE", "VIEW", "SYSTEM TABLE", and "TEMPORARY TABLE". All these properties are read-only as long as this is not a *descriptor*. The descriptor pattern is described later.

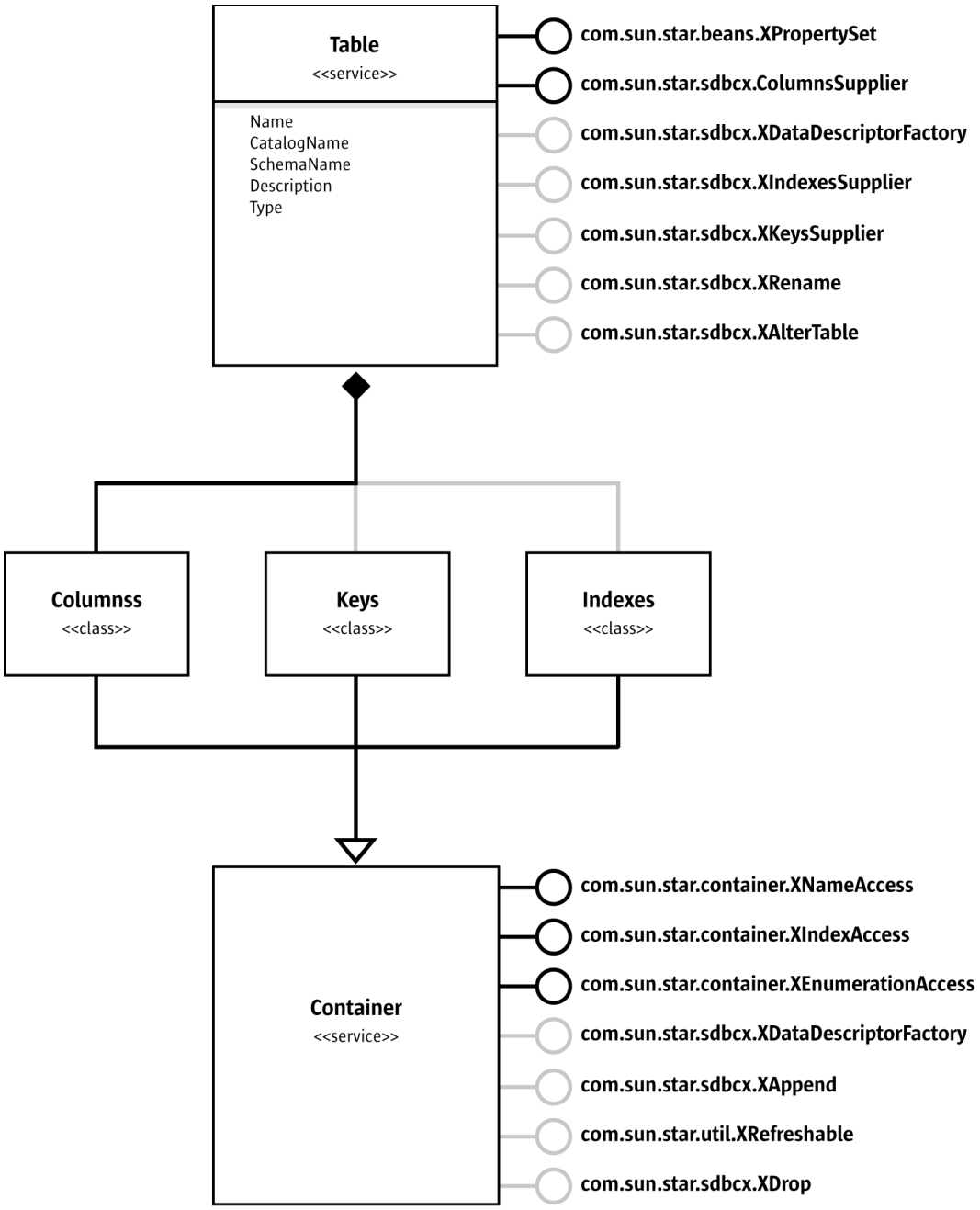


Illustration 184: Table

The `Table` object also supports the `com.sun.star.sdbcx.XColumnsSupplier` interface, because a table can not exist without columns. The other interfaces are optional, that is, they do not have to be supported by the actual table object:

- `com.sun.star.sdbcx.XDataDescriptorFactory` interface that is used to copy a table object.
- `com.sun.star.sdbcx.XIndexesSupplier` interface that returns the container for indexes.
- `com.sun.star.sdbcx.XKeysSupplier` interface that returns the keys container.
- `com.sun.star.sdbcx.XRename` interface that allows renaming a table object.

- `com.sun.star.sdbcx.XAlterTable` interface that allows the altering of columns of a table object.

The code example below shows the use of the table container and prints the table properties of the first table in the container. (Database/sdbcx.java)

...

```
    XNameAccess xTables = xTabSup.getTables();
    if (0 != aTableNames.length) {
        Object table = xTables.getByName(aTableNames[0]);
        XPropertySet xProp = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, table);
        System.out.println("Name:          " + xProp.getPropertyValue("Name"));
        System.out.println("CatalogName:   " + xProp.getPropertyValue("CatalogName"));
        System.out.println("SchemaName:    " + xProp.getPropertyValue("SchemaName"));
        System.out.println("Description:   " + xProp.getPropertyValue("Description"));
        // the following property is optional so we first must check if it exists
        if(xProp.getPropertySetInfo().hasPropertyByName("Type"))
            System.out.println("Type:          " + xProp.getPropertyValue("Type"));
    }
```

The Table object contains access to the columns, keys, and indexes when the above mentioned interfaces are supported. (Database/sdbcx.java)

```
// print all columns of a XColumnsSupplier
// later on used for keys and indexes as well
public static void printColumns(XColumnsSupplier xColumnsSup)
        throws com.sun.star.uno.Exception,SQLException {
    System.out.println("Example printColumns");
    // the table must at least support a XColumnsSupplier interface
    System.out.println("--- Columns ---");
    XNameAccess xColumns = xColumnsSup.getColumns();
    String [] aColumnNames = xColumns.getElementNames();
    for (int i =0; i<= aColumnNames.length-1; i++)
        System.out.println("    " + aColumnNames[i]);
}

// print all keys including the columns of a key
public static void printKeys(XColumnsSupplier xColumnsSup)
        throws com.sun.star.uno.Exception,SQLException {
    System.out.println("Example printKeys");
    XKeysSupplier xKeysSup = (XKeysSupplier)UnoRuntime.queryInterface(
        XKeysSupplier.class, xColumnsSup);
    if (xKeysSup != null) {
        System.out.println("--- Keys ---");
        XIndexAccess xKeys = xKeysSup.getKeys();
        for ( int i =0; i < xKeys.getCount(); i++) {
            Object key = xKeys.getByIndex(i);
            XPropertySet xProp = (XPropertySet)UnoRuntime.queryInterface(
            XPropertySet.class,key);
            System.out.println("    " + xProp.getPropertyValue("Name"));
            XColumnsSupplier xKeyColumnsSup = (XColumnsSupplier)UnoRuntime.queryInterface(
                XColumnsSupplier.class, xProp);
            printColumns(xKeyColumnsSup);
        }
    }
}

// print all indexes including the columns of an index
public static void printIndexes(XColumnsSupplier xColumnsSup)
        throws com.sun.star.uno.Exception,SQLException {
    System.out.println("Example printIndexes");
    XIndexesSupplier xIndexesSup = (XIndexesSupplier)UnoRuntime.queryInterface(
        XIndexesSupplier.class, xColumnsSup);
    if (xIndexesSup != null) {
        System.out.println("--- Indexes ---");
        XNameAccess xIndexs = xIndexesSup.getIndexes();
        String [] aIndexNames = xIndexs.getElementNames();
        for ( int i =0; i<= aIndexNames.length-1; i++) {
            System.out.println("    " + aIndexNames[i]);
            Object index = xIndexs.getByName(aIndexNames[i]);
            XColumnsSupplier xIndexColumnsSup = (XColumnsSupplier)UnoRuntime.queryInterface(
                XColumnsSupplier.class, index);
            printColumns(xIndexColumnsSup);
        }
    }
}
```

Column Service

The Column object is the simplest object structure in the SDBCX layer. It is a collection of properties that define the Column object. The columns container exists for table, key, and index objects. The Column object is a different for these objects:

- The normal `Column` service is used for the table object.
- `com.sun.star.sdbcx.KeyColumn` extends the "normal" `com.sun.star.sdbcx.Column` service with an extra property named `RelatedColumn`. This property is the name of a referenced column out of the referenced table.
- `com.sun.star.sdbcx.IndexColumn` extends the `com.sun.star.sdbcx.Column` service with an extra boolean property named `IsAscending`. This property is true when the index is ascending, otherwise it is false.

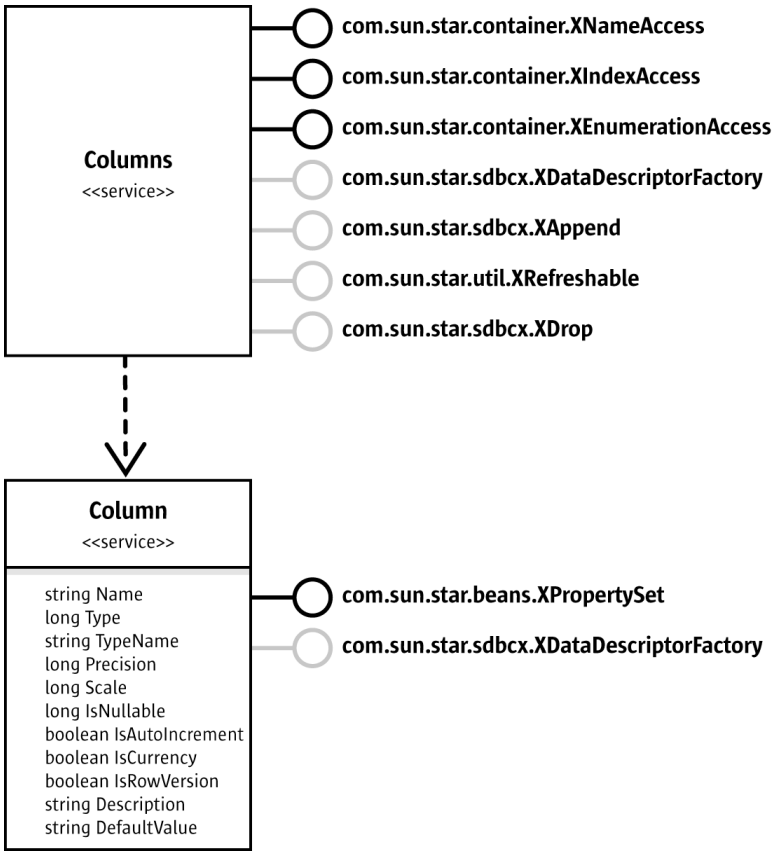


Illustration 185: Column

The Column object is defined by the following properties:

Properties of `com.sun.star.sdbcx.Column`	
`Name`	`string` — The name of the column.
`Type`	`com.sun.star.sdbc.DataType`, `long` — The SDBC data type.
`TypeName`	`string` — The database name for this type.
`Precision`	`long` — The column's number of decimal digits.
`Scale`	`long` — The column's number of digits to the left of the decimal point.

Properties of `com.sun.star.sdbcx.Column`	
`IsNullable`	long —Indicates the nullification of values in the designated column. `com.sun.star.sdbc.ColumnValue`
`IsAutoIncrement`	`boolean` —Indicates if the column is automatically numbered.
`IsCurrency`	`boolean` —Indicates if the column is a cash value.
`IsRowVersion`	`boolean` —Indicates that the column contains some kind of time or date stamp used to track updates (optional).
`Description`	`string` —Keeps a description of the object (optional).
`DefaultValue`	`string` —Keeps a default value for a column (optional).

The Column object also supports the `com.sun.star.sdbcx.XDataDescriptorFactory` interface that creates a copy of this object. (Database/sdbcx.java)

```
// column properties
public static void printColumnProperties(Object column) throws com.sun.star.uno.Exception,SQLException {
        System.out.println("Example printColumnProperties");
        XPropertySet xProp = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class,column);
        System.out.println("Name:            " + xProp.getPropertyValue("Name"));
        System.out.println("Type:            " + xProp.getPropertyValue("Type"));
        System.out.println("TypeName:        " + xProp.getPropertyValue("TypeName"));
        System.out.println("Precision:       " + xProp.getPropertyValue("Precision"));
        System.out.println("Scale:           " + xProp.getPropertyValue("Scale"));
        System.out.println("IsNullable:      " + xProp.getPropertyValue("IsNullable"));
        System.out.println("IsAutoIncrement: " + xProp.getPropertyValue("IsAutoIncrement"));
        System.out.println("IsCurrency:      " + xProp.getPropertyValue("IsCurrency"));
        // the following property is optional so we first must check if it exists
        if(xProp.getPropertySetInfo().hasPropertyByName("IsRowVersion"))
                System.out.println("IsRowVersion:    " + xProp.getPropertyValue("IsRowVersion"));
        if(xProp.getPropertySetInfo().hasPropertyByName("Description"))
                System.out.println("Description:     " + xProp.getPropertyValue("Description"));
        if(xProp.getPropertySetInfo().hasPropertyByName("DefaultValue"))
                System.out.println("DefaultValue:    " + xProp.getPropertyValue("DefaultValue"));
}
```

Index Service

The Index service encapsulates indexes at a table object. An index is described through the properties `Name`, `Catalog`, `IsUnique`, `IsPrimaryKeyIndex`, and `IsClustered`. All properties are read-only if an index has not been added to a tables index container. The last three properties are boolean values that indicate an index object only allows unique values, is used for the primary key, and if it is clustered. The property `IsPrimaryKeyIndex` is only available after the index has been created because it defines a special index that is created by the database while creating a primary key for a table object. Not all databases currently available in OpenOffice.org API support primary keys.

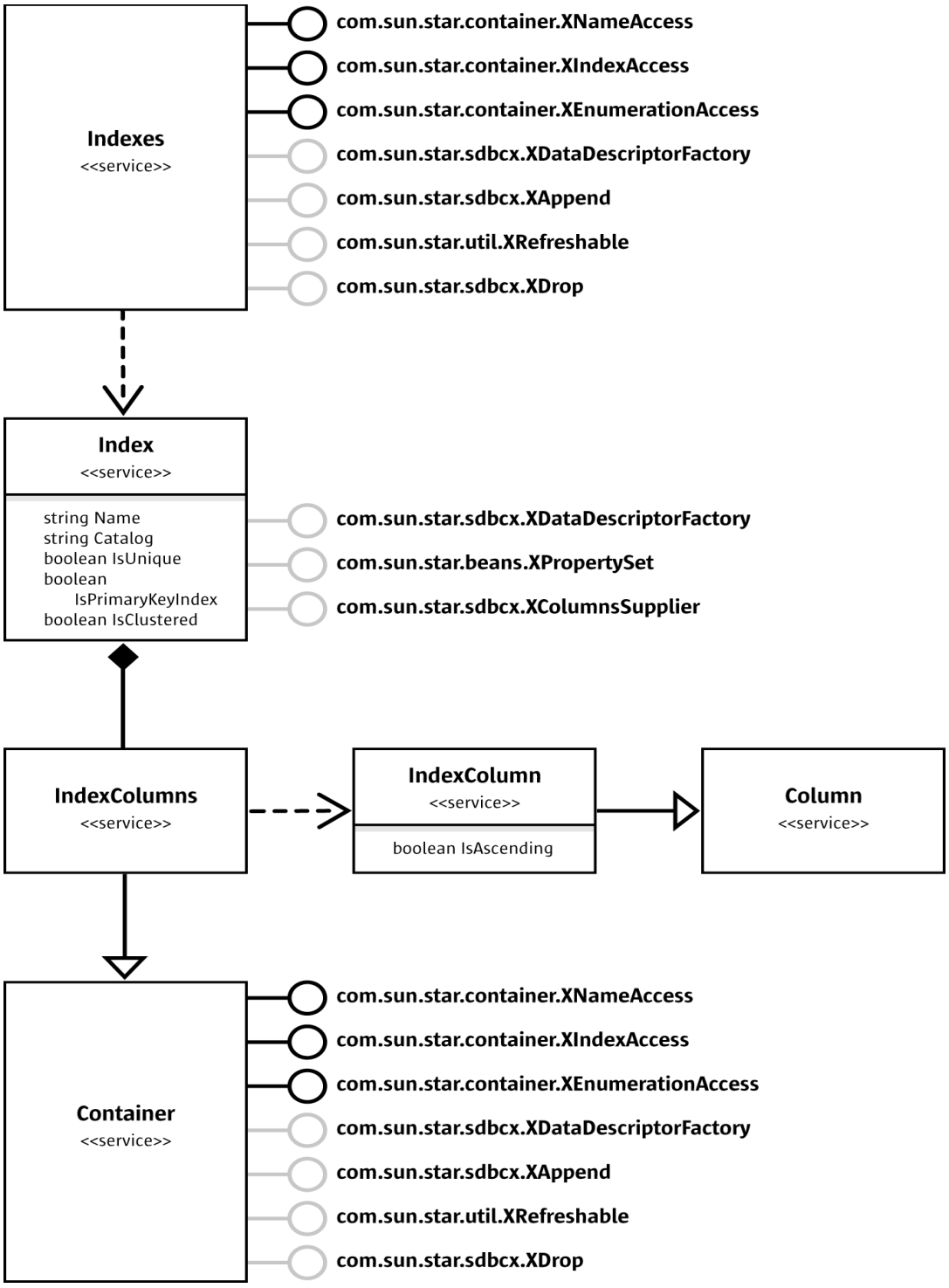


Illustration 186: Index

The following code fragment displays the properties of a given index object: (Database/sdbcx.java)

```
// index properties
public static void printIndexProperties(Object index) throws Exception, SQLException {
    System.out.println("Example printIndexProperties");
    XPropertySet xProp = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, index);
    System.out.println("Name:              " + xProp.getPropertyValue("Name"));
    System.out.println("Catalog:           " + xProp.getPropertyValue("Catalog"));
```

```
    System.out.println("IsUnique:          " + xProp.getPropertyValue("IsUnique"));
    System.out.println("IsPrimaryKeyIndex: " + xProp.getPropertyValue("IsPrimaryKeyIndex"));
    System.out.println("IsClustered:       " + xProp.getPropertyValue("IsClustered"));
}
```

Key Service

The Key service provides the foreign and primary keys behavior through the following properties. The `Name` property is the name of the key. It could happen that the primary key does not have a name. The property `Type` contains the kind of the key, that could be PRIMARY, UNIQUE, or FOREIGN, as specified by the constant group `com.sun.star.sdbcx.KeyType`. The property `ReferencedTable` contains a value when the key is a foreign key and it designates the table to which a foreign key points. The `DeleteRule` and `UpdateRule` properties determine what happens when a primary key is deleted or updated. The possibilities are defined in `com.sun.star.sdbc.KeyRule`: CASCADE, RESTRICT, SET_NULL, NO_ACTION and SET_DEFAULT.

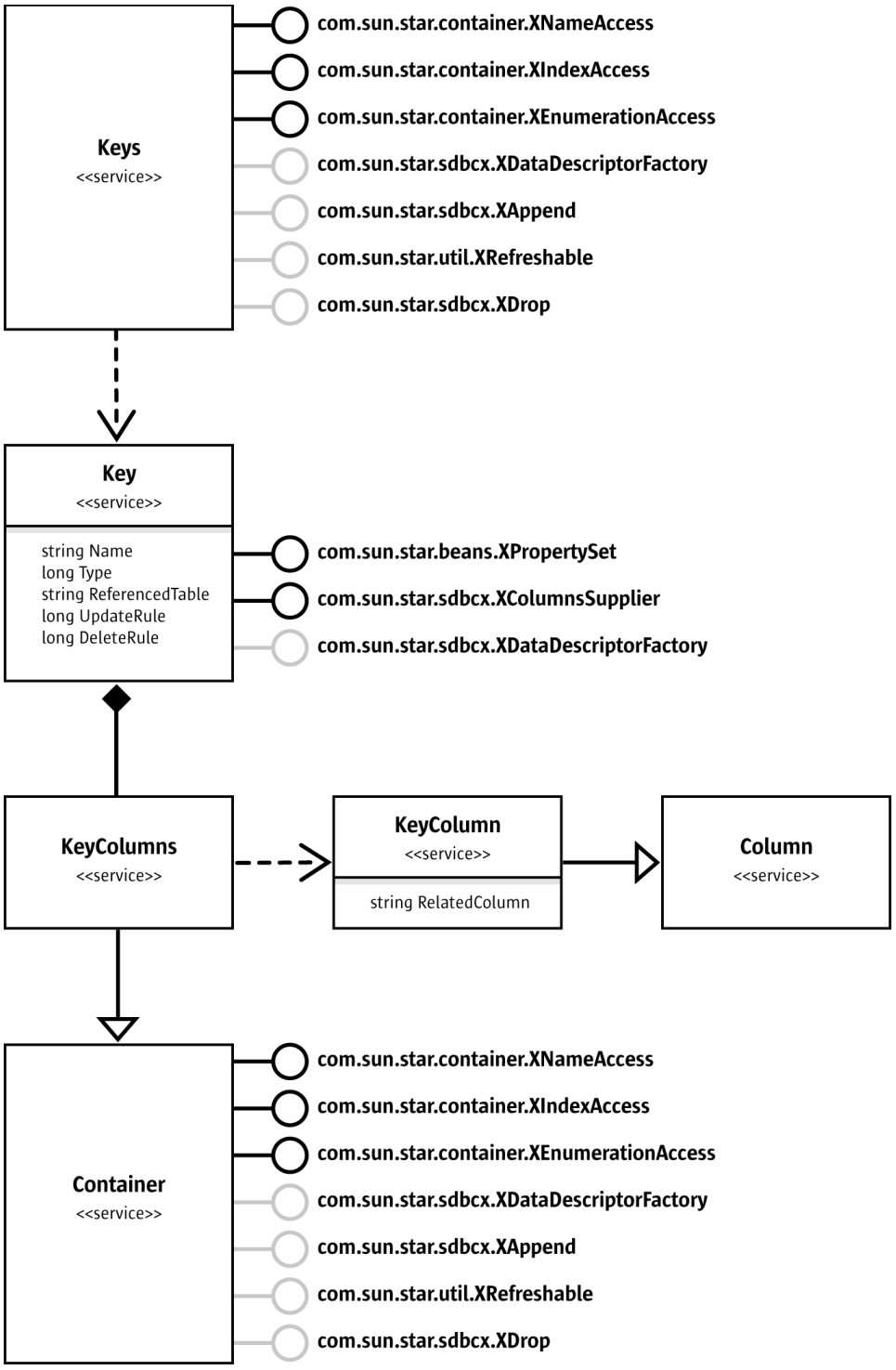


Illustration 187: Key

The following code fragment displays the properties of a given key object: (Database/sdbcx.java)

// key properties

```
public static void printKeyProperties(Object key) throws Exception, SQLException {
    System.out.println("Example printKeyProperties");
    XPropertySet xProp = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, key);
    System.out.println("Name:            " + xProp.getPropertyValue("Name"));
    System.out.println("Type:            " + xProp.getPropertyValue("Type"));
```

```
    System.out.println("ReferencedTable: " + xProp.getPropertyValue("ReferencedTable"));
    System.out.println("UpdateRule:      " + xProp.getPropertyValue("UpdateRule"));
    System.out.println("DeleteRule:      " + xProp.getPropertyValue("DeleteRule"));
}
```

View Service

A view is a virtual table created from a SELECT on other database tables or views. This service creates a database view programmatically. It is not necessary to know the SQL syntax for the CREATE VIEW statement, but a few properties have to be set. When creating a view, supply the value for the property `Name`, the SELECT statement to the property `Command` and if the database driver supports a check option, set it in the property `CheckOption`. Possible values of `com.sun.star.sdbcx.CheckOption` are NONE, CASCADE and LOCAL. A schema or catalog name can be provided (optional).

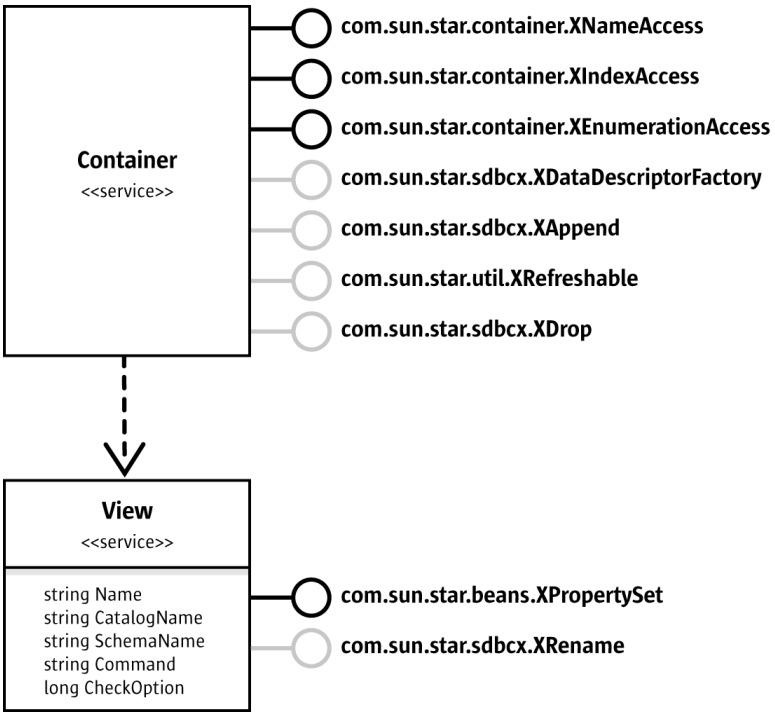


Illustration 188: View

Group Service

The service `[idl:com.sun.star.sdbcx.Group]` is the first of the two security services, `Group` and `User`. The `Group` service represents the group account that has access permissions to a secured database and it has a `Name` property to identify it. It supports the interface `com.sun.star.sdbcx.XAuthorizable` that allows current privilege settings to be obtained, and to grant or revoke privileges. The second interface is the `com.sun.star.sdbcx.XUsersSupplier`. The word 'Supplier' in the interface name identifies the group object as a container for users. The container returned here is a collection of all users that belong to this group.

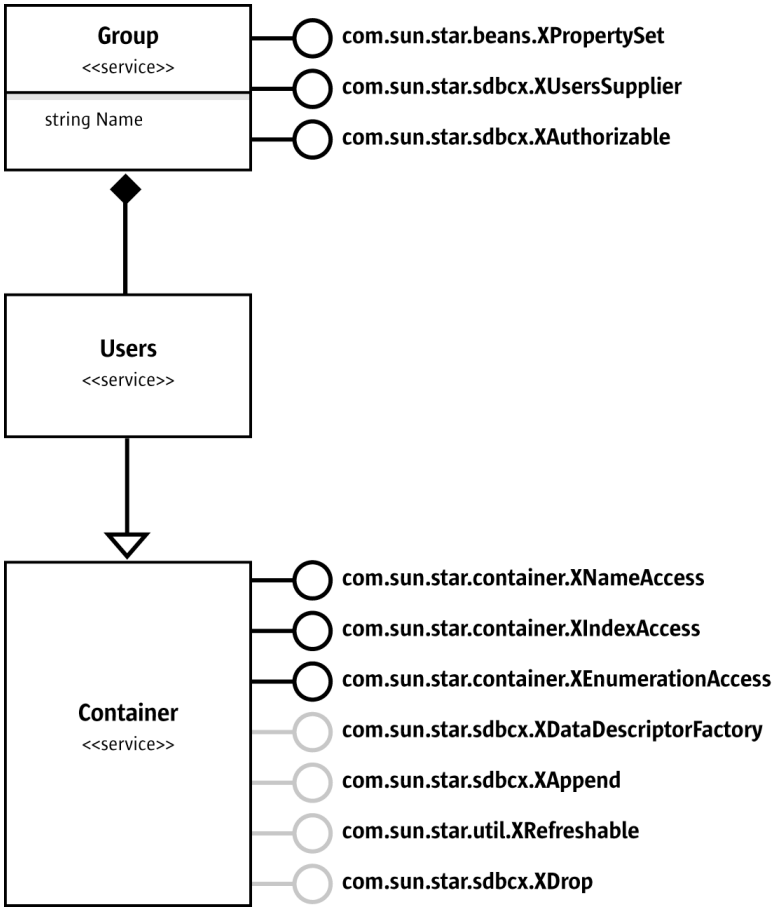


Illustration 189: Group

(Database/sdbcx.java)

```
// print all groups and the users with their privileges who belong to this group
public static void printGroups(XTablesSupplier xTabSup) throws com.sun.star.uno.Exception, SQLException
{
    System.out.println("Example printGroups");
    XGroupsSupplier xGroupsSup = (XGroupsSupplier)UnoRuntime.queryInterface(
        XGroupsSupplier.class, xTabSup);
    if (xGroupsSup != null) {
        // the table must be at least support a XColumnsSupplier interface
        System.out.println("--- Groups ---");
        XNameAccess xGroups = xGroupsSup.getGroups();
        String [] aGroupNames = xGroups.getElementNames();
        for (int i =0; i < aGroupNames.length; i++) {
            System.out.println("    " + aGroupNames[i]);
            XUsersSupplier xUsersSup = (XUsersSupplier)UnoRuntime.queryInterface(
                                XUsersSupplier.class, xGroups.getByName(aGroupNames[i]));
            if (xUsersSup != null) {
                XAuthorizable xAuth = (XAuthorizable)UnoRuntime.queryInterface(
                    XAuthorizable.class, xUsersSup);
                // the table must be at least support a XColumnsSupplier interface
                System.out.println("\t--- Users ---");
                XNameAccess xUsers = xUsersSup.getUsers();
                String [] aUserNames = xUsers.getElementNames();
                for (int j = 0; j < aUserNames.length; j++) {
                    System.out.println("\t    " + aUserNames[j] +
                    " Privileges: " + xAuth.getPrivileges(aUserNames[j], PrivilegeObject.TABLE));
                }
            }
        }
    }
}
```

User Service

The `[idl:com.sun.star.sdbcx.User]` service is the second security service, representing a user in the catalog. This object has the property Name that is the user name. Similar to the `Group` service, the `User` service supports the interface `com.sun.star.sdbcx.XAuthorizable`. This is achieved through the interface `com.sun.star.sdbcx.XUser` derived from `XAuthorizable`. In addition to this interface, the `XUser` interface supports changing the password of a specific user. Similar to the `Group` service above, the `User` service is a container for the groups the user belongs to.

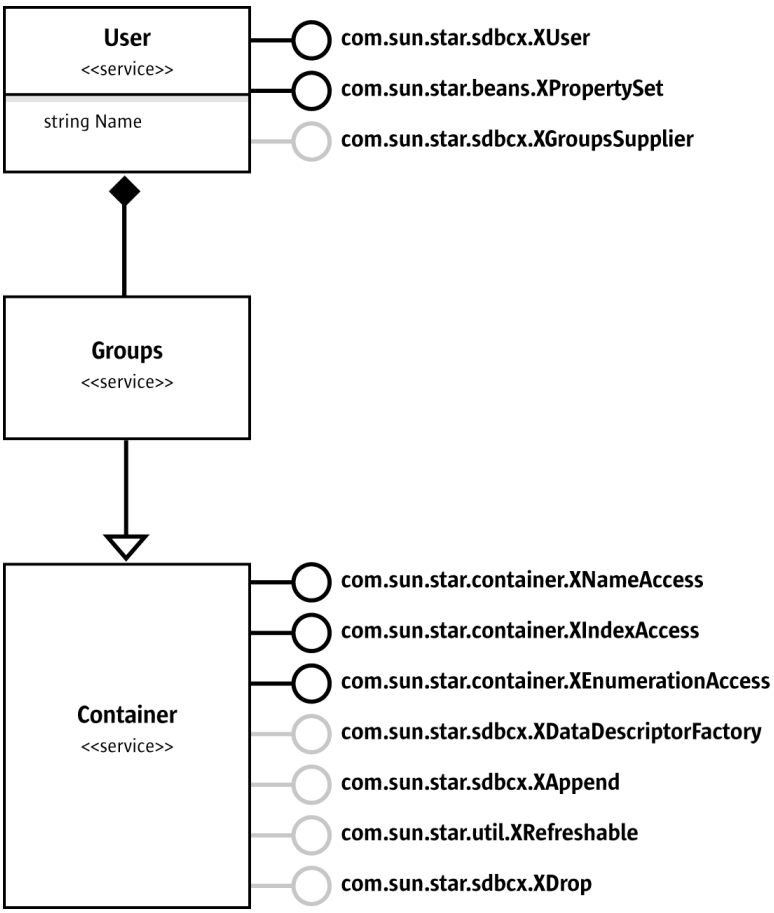


Illustration 190: User

The Descriptor Pattern

The descriptor is a special kind of object that mirrors the structure of the object which should be appended to a container object. This means that a descriptor, once created, can be appended more than once with only small changes to the structure. For example, when appending columns to the columns container, we:

- Create one descriptor with `com.sun.star.sdbcx.XDataDescriptorFactory`.
- Set the needed properties.
- Add the descriptor to the container.
- Adjust some properties, such as the name.

- Add the modified descriptor to the container.
- Repeat the steps, as necessary.

therefore, only create one descriptor to append more than one column.

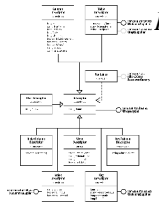


Illustration 191: Descriptor Pattern

- Creating a Table

An important use of the SDBCX layer is that it is possible to programmatically create tables, along with their columns, indexes, and keys.

The method of creating a table is the same as creating a table with a graphical table design. To create it programmatically is easy. First, create a table object by asking the tables container for its `com.sun.star.sdbcx.XDataDescriptorFactory` interface. When the `createDataDescriptor` method is called, the `com.sun.star.beans.XPropertySet` interface of an object that implements the service `com.sun.star.sdbcx.TableDescriptor` is returned. As described above, use this descriptor to create a new table in the database, by adding the descriptor to the Tables container. Before appending the descriptor, append the columns to the table descriptor. Use the same method as with the containers used in the SDBCX layer. On the column object, some properties need to be set, such as `Name`, and `Type`. The properties to be set depend on the SDBC data type of the column.

The column name must be unique in the columns container.

After the columns are appended, add the `TableDescriptor` object to its container or define some key objects, such as a primary key. (Database/sdbcx.java)

```
// create the table salesmen
public static void createTableSalesMen(XNameAccess xTables) throws Exception, SQLException {
    XDataDescriptorFactory xTabFac = (XDataDescriptorFactory)UnoRuntime.queryInterface(
        XDataDescriptorFactory.class, xTables);

    if (xTabFac != null) {
        // create the new table
        XPropertySet xTable = xTabFac.createDataDescriptor();
        // set the name of the new table
        xTable.setPropertyValue("Name", "SALESMAN");

        // append the columns
        XColumnsSupplier xColumSup = (XColumnsSupplier)UnoRuntime.queryInterface(
            XColumnsSupplier.class,xTable);
        XDataDescriptorFactory xColFac = (XDataDescriptorFactory)UnoRuntime.queryInterface(
            XDataDescriptorFactory.class, xColumSup.getColumns());
        XAppend xAppend = (XAppend)UnoRuntime.queryInterface(XAppend.class, xColFac);

        // we only need one descriptor
        XPropertySet xCol = xColFac.createDataDescriptor();

        // create first column and append
        xCol.setPropertyValue("Name", "SNR");
        xCol.setPropertyValue("Type", new Integer(DataType.INTEGER));
        xCol.setPropertyValue("IsNullable", new Integer(ColumnValue.NO_NULLS));
        xAppend.appendByDescriptor(xCol);
        // 2nd only set the properties which differ
        xCol.setPropertyValue("Name", "FIRSTNAME");
        xCol.setPropertyValue("Type", new Integer(DataType.VARCHAR));
        xCol.setPropertyValue("IsNullable", new Integer(ColumnValue.NULLABLE));
        xCol.setPropertyValue("Precision", new Integer(50));
        xAppend.appendByDescriptor(xCol);
        // 3rd only set the properties which differ
        xCol.setPropertyValue("Name", "LASTNAME");
        xCol.setPropertyValue("Precision", new Integer(100));
        xAppend.appendByDescriptor(xCol);
        // 4th only set the properties which differ
```

```
        xCol.setPropertyValue("Name", "STREET");
        xCol.setPropertyValue("Precision",n ew Integer(50));
        xAppend.appendByDescriptor(xCol);
        // 5th only set the properties which differ
        xCol.setPropertyValue("Name", "STATE");
        xAppend.appendByDescriptor(xCol);
        // 6th only set the properties which differ
        xCol.setPropertyValue("Name", "ZIP");
        xCol.setPropertyValue("Type", new Integer(DataType.INTEGER));
        xCol.setPropertyValue("Precision", new Integer(10)); // default value integer
        xAppend.appendByDescriptor(xCol);
        // 7th only set the properties which differs
        xCol.setPropertyValue("Name", "BIRTHDATE");
        xCol.setPropertyValue("Type", new Integer(DataType.DATE));
        xCol.setPropertyValue("Precision", new Integer(10)); // default value integer
        xAppend.appendByDescriptor(xCol);

        // now we create the primary key
        XKeysSupplier xKeySup = (XKeysSupplier)UnoRuntime.queryInterface(XKeysSupplier.class, xTable);
        XDataDescriptorFactory xKeyFac = (XDataDescriptorFactory)UnoRuntime.queryInterface(
            XDataDescriptorFactory.class,xKeySup.getKeys());
        XAppend xKeyAppend = (XAppend)UnoRuntime.queryInterface(XAppend.class, xKeyFac);

        XPropertySet xKey = xKeyFac.createDataDescriptor();
        xKey.setPropertyValue("Type", new Integer(KeyType.PRIMARY));
        // now append the columns to key
        XColumnsSupplier xKeyColumSup = (XColumnsSupplier)UnoRuntime.queryInterface(
            XColumnsSupplier.class, xKey);
        XDataDescriptorFactory xKeyColFac = (XDataDescriptorFactory)UnoRuntime.queryInterface(
            XDataDescriptorFactory.class,xKeyColumSup.getColumns());
        XAppend xKeyColAppend = (XAppend)UnoRuntime.queryInterface(XAppend.class, xKeyColFac);

        // we only need one descriptor
        XPropertySet xKeyCol = xKeyColFac.createDataDescriptor();
        xKeyCol.setPropertyValue("Name", "SNR");
        // append the key column
        xKeyColAppend.appendByDescriptor(xKeyCol);
        // append the key
        xKeyAppend.appendByDescriptor(xKey);
        // the last step is to append the new table to the tables collection
        XAppend xTableAppend = (XAppend)UnoRuntime.queryInterface(XAppend.class, xTabFac);
        xTableAppend.appendByDescriptor(xTable);
    }
}
```

Adding an Index

To add an index, the same programmatic logic is followed. Create an `IndexDescriptor` with the `com.sun.star.sdbcx.XDataDescriptorFactory` interface from the index container. Then follow the same steps as for the table. Next, append the columns to be indexed.

Note that only an index can be added to an existing table. It is not possible to add an index to a TableDescriptor.

The task is completed when the index object is added to the index container, unless the `append()` method throws an `com.sun.star.sdbc.SQLException`. This may happen when adding a unique index on a column that already contains values that are not unique.+

Creating a User

The procedure to create a user is the same. The `com.sun.star.sdbcx.XDataDescriptorFactory` interface is used from the users container. Create a user with the `UserDescriptor`. The `com.sun.star.sdbcx.UserDescriptor` has an additional property than the `User` service supports. This additional property is the `Password` property which should be set. Then the `UserDescriptor` object can be appended to the user container. (Database/sdbcx.java)

```
// create a user
public static void createUser(XNameAccess xUsers) throws Exception,SQLException {
    System.out.println("Example createUser");
    XDataDescriptorFactory xUserFac = (XDataDescriptorFactory)UnoRuntime.queryInterface(
        XDataDescriptorFactory.class, xUsers);
    if (xUserFac != null) {
```

```
        // create the new table
        XPropertySet xUser = xUserFac.createDataDescriptor();
        // set the name of the new table
        xUser.setPropertyValue("Name", "BOSS");
        xUser.setPropertyValue("Password","BOSSWIFENAME");
        XAppend xAppend = (XAppend)UnoRuntime.queryInterface(XAppend.class, xUserFac);
        xAppend.appendByDescriptor(xUser);
    }
}
```

Adding a Group

Creating a `com.sun.star.sdbcx.GroupDescriptor` object is the same as the methods described above. Follow the same steps:

1. Set a name for the group in the `Name` property.
2. Append all the users to the user container of the group.
3. Append the `GroupDescriptor` object to the group container of the catalog.

12.5 Using DBMS Features

12.5.1 Transaction Handling

Transactions combine several separate SQL executions, so that they can be seen as a single event that is executed completely (commit) or not at all (rollback). A typical example for a transaction is a money transfer. It consists of two steps: withdrawing an amount of money from one bank account and crediting another account with it. Both steps must be successful or they must be canceled. Transactions in SDBC are handled by the `com.sun.star.sdbc.XConnection` interface of connections. The transaction related methods of this interface are:

```
// transactions
void setTransactionIsolation( [in] long level)
long getTransactionIsolation()
void setAutoCommit( [in] boolean autoCommit)
boolean getAutoCommit()
void commit()
void rollback()
```

Usually all transactions are in auto commit mode, that means, a commit takes place after each single SQL command. Therefore to control a transaction manually, switch auto commit off using `setAutoCommit(false)`. The first SQL command without auto commit starts a transaction that is active until the corresponding methods have been committed or rolled back.

Afterwards, the auto commit mode can be reinstated using `setAutoCommit(true)`.

Transactions bring about a synchronization problem. If data is read from a table, it is possible that the data has just been changed by a command of a transaction started by another process. If the other transaction is rolled back, there may be inconsistencies between the results and contents of the database.

Transaction isolation controls the behavior of the database in case of parallel transactions. There are several isolation levels:

Values of constants `com.sun.star.sdbc.TransactionIsolation`	
NONE	Indicates that transactions are not supported.

Values of constants `com.sun.star.sdbc.TransactionIsolation`	
READ_UNCOMMITTED	Dirty reads, non-repeatable reads and phantom reads can occur. This level allows a row changed by one transaction to be read by another transaction before any changes in that row have been committed (a "dirty read"). If any of the changes are rolled back, the second transaction retrieves an invalid row.
[IDLs:com.sun.star.sdbc.TransactionIsolation:READ_COMMITTED]	Dirty reads are prevented; non-repeatable reads and phantom reads can occur. This level only prohibits a transaction from reading a row with uncommitted changes in it.
REPEATABLE_READ	Dirty reads and non-repeatable reads are prevented; phantom reads can occur. This level prohibits a transaction from reading a row with uncommitted changes in it, and it also prohibits the situation where one transaction reads a row, a second transaction alters the row, and the first transaction rereads the row, getting different values the second time (a "non-repeatable read").
SERIALIZABLE	Dirty reads, non-repeatable reads and phantom reads are prevented. This level includes the prohibitions in REPEATABLE_READ and further prohibits the situation where one transaction reads all rows that satisfy a WHERE condition, a second transaction inserts a row that satisfies that WHERE condition, and the first transaction rereads for the same condition, retrieving the additional "phantom" row in the second read.

12.5.2 Stored Procedures

Stored procedures are server-side processes execute several SQL commands in a single step, and can be embedded in a server language for stored procedures with enhanced control capabilities. A procedure call usually has to be parameterized, and the results are result sets and additional out parameters. Stored procedures are handled by the method `prepareCall()` of the interface `com.sun.star.sdbc.XConnection`.

```
com::sun::star::sdbc::XPreparedStatement prepareCall( [in] string sql)
```

The method `prepareCall()` takes a an SQL statement that may contain one or more '?' in parameter placeholders. It returns a `com.sun.star.sdbc.CallableStatement`. A `CallableStatement` is a `com.sun.star.sdbcx.PreparedStatement` with two additional interfaces for out parameters:

`com.sun.star.sdbc.XOutParameters` is used to declare parameters as out parameters. All out parameters must be registered before a stored procedure is executed.

Methods of `com.sun.star.sdbc.XOutParameters`	
`registerOutParameter()`	Takes the arguments long parameterIndex, long sqlType, string typeName. Registers an output parameter and should be used for a user-named or REF output parameter. Examples of user-named types include: STRUCT, DISTINCT, OBJECT, and named array types.
`registerNumericOutParameter()`	Takes the arguments long parameterIndex, long sqlType, long scale. Registers an out parameter in the ordinal position parameterIndex with the `com.sun.star.sdbc.DataType` sqlType; scale is the number of digits on the right-hand side of the decimal point.

The `com.sun.star.sdbc.XRow` is used to retrieve the values of out parameters. It consists of `getXXX()` methods and should be well-known from the common result sets.

12.6 Writing Database Drivers

In the following sections, implementing an SDBC driver is described. The user should have some experience in the use of the SDBC API, or be familiar with the previous chapter about SDBC and SDBCX.

This section is divided into two parts. The first part describes the simple driver that includes only the SDBC layer with the PreparedStatements, Statements and ResultSets. The second part extends the simple driver from part one to a more sophisticated one. This driver provides access to Tables, Views, Groups, Users and others.

A skeleton for a C++ SDBC driver is provided in the samples folder. Some changes are necessary to create a working driver. Adjust the namespace and replace the word "skeleton" by a suitable driver name, and implement the necessary functions for the database.

An SDBC driver is simply the implementation of some SDBC services previously discussed.

12.6.1 SDBC Driver

The SDBC driver consists of seven services. Each service needs to be defined and are described in the next sections. Below is a list of all the services that define the driver:

- `Driver`, a singleton which creates the connection object.
- `Connection`, creates `Statement`, `PreparedStatement` and gives access to the `DatabaseMetaData`.
- `DatabaseMetaData`, returns information about the used database.
- `Statement`, creates `ResultSets`.
- `PreparedStatement`, creates `ResultSets` in conjunction with parameters.
- `ResultSet`, fetches the data returned by an SQL statement.
- `ResultSetMetaData`, describes the columns of a `ResultSet`.

The relationship between these services is depicted in Illustration 170.

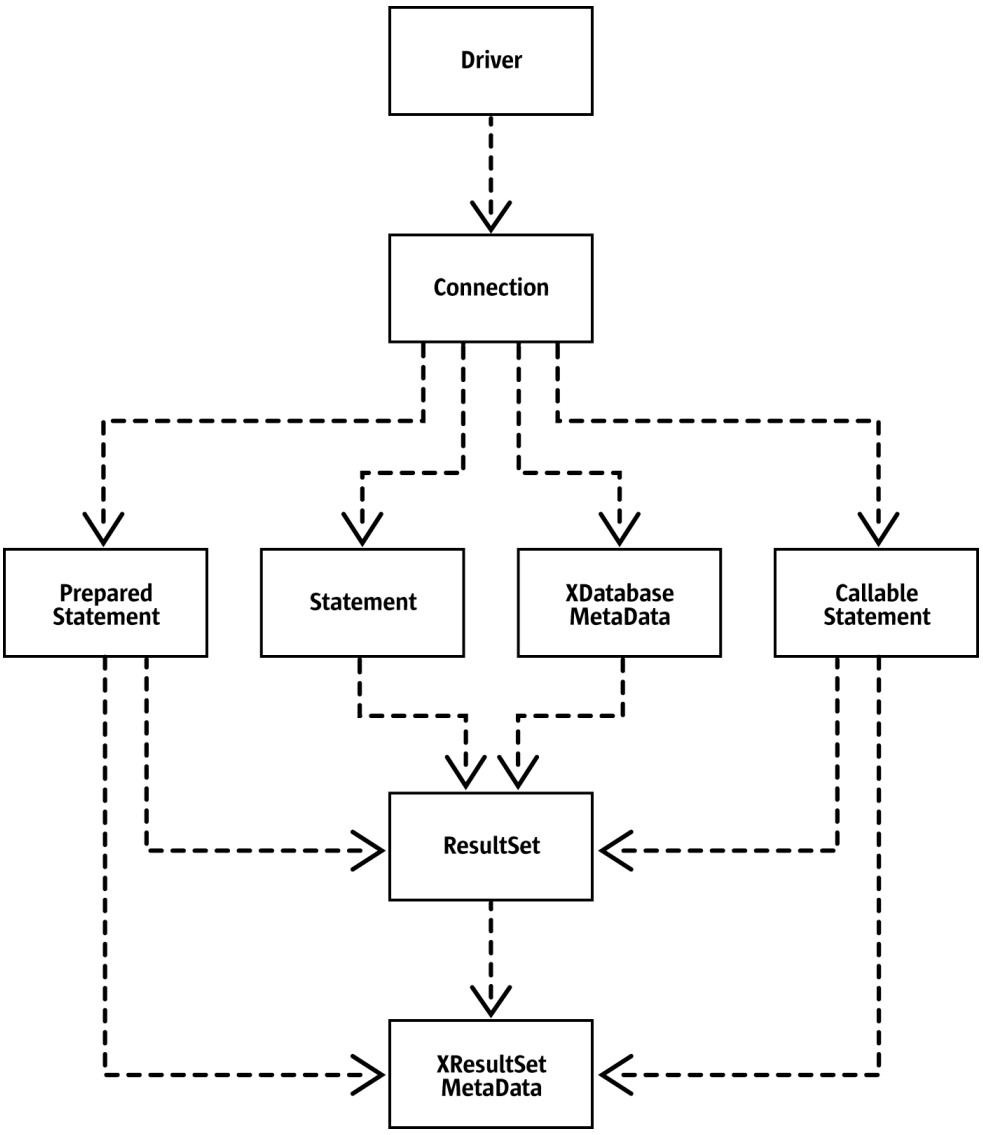


Illustration 192: Dependency between driver classes

12.6.2 Driver Service

The `Driver` service is the entry point to create the first contact with any database. As shown in the illustration above, the class that implements the service `Driver` is responsible for creating a connection object that represents the database on the client side.

The class must be derived from the interface `com.sun.star.sdbc.XDriver` that defines the methods needed to create a connection object. The code in the following lines shows a snippet of a driver class. (Database/DriverSkeleton/SDriver.cxx)

```
// ------------------------------------------------------------------------------
Reference< XConnection > SAL_CALL SkeletonDriver::connect( const ::rtl::OUString& url,
                const Sequence< PropertyValue >& info ) throw(SQLException, RuntimeException)
{
        // create a new connection with the given properties and append it to our vector
        OConnection* pCon = new OConnection(this);
        Reference< XConnection > xCon = pCon;     // important here because otherwise the connection
                                                  // could be deleted inside (refcount goes -> 0)
        pCon->construct(url,info);                // late constructor call which can throw exception
```

```
                                                    // and allows a correct dtor call when so
        m_xConnections.push_back(WeakReferenceHelper(*pCon));

        return xCon;
}
// ------------------------------------------------------------------------------
sal_Bool SAL_CALL SkeletonDriver::acceptsURL( const ::rtl::OUString& url )
                throw(SQLException, RuntimeException)
{
        // here we have to look if we support this url format
        // change the URL format to your needs, but please be aware that
        //the first who accepts the URL wins.
        return (!url.compareTo(::rtl::OUString::createFromAscii("sdbc:skeleton:"),14));
}
// ------------------------------------------------------------------------------
Sequence< DriverPropertyInfo > SAL_CALL SkeletonDriver::getPropertyInfo( const ::rtl::OUString& url,
                const Sequence< PropertyValue >& info ) throw(SQLException, RuntimeException)
{
        // if you have something special to say, return it here :-)
        return Sequence< DriverPropertyInfo >();
}
// ------------------------------------------------------------------------------
sal_Int32 SAL_CALL SkeletonDriver::getMajorVersion(  ) throw(RuntimeException)
{
        return 0; // depends on you
}
// ------------------------------------------------------------------------------
sal_Int32 SAL_CALL SkeletonDriver::getMinorVersion(  ) throw(RuntimeException)
{
        return 1; // depends on you
}
// ------------------------------------------------------------------------------
```

The main methods of this class are acceptsURL and connect:

- The method `acceptsURL()` is called every time a user wants to create a connection through the `DriverManager`, because the `DriverManager` decides the Driver it should ask to connect to the given URL. Therefore this method should be small and run very fast.
- The method `connect()` is called after the method `acceptsURL()` is invoked and returned true. The `connect()` could be seen as a factory method that creates `Connection` services specific for a driver implementation. To accomplish this, the `Driver` class must be singleton. Singleton means that only one instance of the `Driver` class may exist at the same time.

If more information is required about the other methods, refer to `com.sun.star.sdbc.Driver` for a complete description.

12.6.3 Connection Service

The `com.sun.star.sdbc.Connection` is the database client side. It is responsible for the creation of the Statements and the information about the database itself. The service consists of three interfaces that have to be supported:

- The interface `com.sun.star.lang.XComponent` that is responsible to close the connection when it is disposed.
- The interface `com.sun.star.sdbc.XWarningsSupplier` that controls the chaining of warnings which may occur on every call.
- The interface `com.sun.star.sdbc.XConnection` that is the main interface to the database.

The first two interfaces introduce some access and closing mechanisms that can be best described inside the code fragment of the `Connection` class. To understand the interface `com.sun.star.sdbc.XConnection`, we must have a closer look at some methods. The others not described are simple enough to handle them in the code fragment.

First there is the method `getMetaData()` that returns an object which implements the interface `com.sun.star.sdbc.XDatabaseMetaData`. This object has many methods and depends on the

capabilities of the database. Most return values are found in the database documentation or in the first step, assuming some values match. The methods, such as `getTables()`, `getColumns()` and `getTypeInfo()` are described in the next chapter.

The following methods are used to create statements. Each of them is a factory method that creates the three different kinds of statements.

Important Methods of `com.sun.star.sdbc.XConnection`	
`createStatement()`	Creates a new `com.sun.star.sdbc.Statement` object for sending SQL statements to the database. SQL statements without parameters are executed using Statement objects.
`prepareStatement(sql)`	Creates a `com.sun.star.sdbc.PreparedStatement` object for sending parameterized SQL statements to the database.
`prepareCall(sql)`	Creates a `com.sun.star.sdbc.CallableStatement` object for calling database stored procedures.

(Database/DriverSkeleton/SDriver.cxx)

```
Reference< XStatement > SAL_CALL OConnection::createStatement(  ) throw(SQLException, RuntimeException)
{
        ::osl::MutexGuard aGuard( m_aMutex );
        checkDisposed(OConnection_BASE::rBHelper.bDisposed);

        // create a statement
        // the statement can only be executed once
        Reference< XStatement > xReturn = new OStatement(this);
        m_aStatements.push_back(WeakReferenceHelper(xReturn));
        return xReturn;
}
// -------------------------------------------------------------------------------
Reference< XPreparedStatement > SAL_CALL OConnection::prepareStatement( const ::rtl::OUString& _sSql )
                throw(SQLException, RuntimeException)
{
        ::osl::MutexGuard aGuard( m_aMutex );
        checkDisposed(OConnection_BASE::rBHelper.bDisposed);

        // the pre
        if(m_aTypeInfo.empty())
                buildTypeInfo();

        // create a statement
        // the statement can only be executed more than once
        Reference< XPreparedStatement > xReturn = new OPreparedStatement(this,m_aTypeInfo,_sSql);
        m_aStatements.push_back(WeakReferenceHelper(xReturn));
        return xReturn;
}
// -------------------------------------------------------------------------------
Reference< XPreparedStatement > SAL_CALL OConnection::prepareCall( const ::rtl::OUString& _sSql )
                throw(SQLException, RuntimeException)
{
        ::osl::MutexGuard aGuard( m_aMutex );
        checkDisposed(OConnection_BASE::rBHelper.bDisposed);

        // not implemented yet :-) a task to do
        return NULL;
}
```

All other methods can be omitted at this stage. For detailed descriptions, refer to the API Reference Manual.

12.6.4 XDatabaseMetaData Interface

The `com.sun.star.sdbc.XDatabaseMetaData` interface is the largest interface existing in the SDBC API. This interface knows everything about the used database. It provides information, such as the available tables with their columns, keys and indexes, and information about identifiers that should be used. This chapter explains some of the methods that are frequently used and how they are used to achieve a robust `Driver`.

Important Methods of `com.sun.star.sdbc.XDatabaseMetaData`	
`isReadOnly()`	Returns the state of the database. When true, the database is not editable later in OpenOffice.org API.
`usesLocalFiles()`	Returns true when the catalog name of the database should not appear in the DatasourceBrowser of OpenOffice.org API, otherwise false is returned.
`supportsMixedCaseQuotedIdenti-fiers()`	When this method returns true,the quoted identifiers are case sensitive. For example, in a driver that supports mixed case quoted identifiers, SELECT * FROM "MyTable" retrieves data from a table with the case-sensitive name MyTable.
`getTables()`	Returns a `ResultSet` object that returns a single row for each table that fits the search criteria, such as the catalog name, schema pattern, table name pattern and sequence of table types. The correct column count and names of the columns are found at `com.sun.star.sdbc.XDatabaseMetaData:getTables()`. If this method does not return any rows, this driver does not work with OpenOffice.org API.

Any other `getXXX()` method can be implemented step by step. For the the first step they return an empty `ResultSet` object that contains no rows. It is not allowed to return `NULL` here.

The skeleton driver defines empty `ResultSet`s for these get methods. (Database/DriverSkeleton/SDriver.cxx)

```
Reference< XResultSet > SAL_CALL ODatabaseMetaData::getTables(
                const Any& catalog, const ::rtl::OUString& schemaPattern,
                const ::rtl::OUString& tableNamePattern, const Sequence< ::rtl::OUString >& types )
                throw(SQLException, RuntimeException)
{
        // this returns an empty resultset where the column-names are already set
        // in special the metadata of the resultset already returns the right columns
        ODatabaseMetaDataResultSet* pResultSet = new ODatabaseMetaDataResultSet();
        Reference< XResultSet > xResultSet = pResultSet;
        pResultSet->setTablesMap();
        return xResultSet;
}
```

12.6.5 Statements

Statements are used to create `ResultSet`s or to update the database. The `executeQuery()` method creates new `ResultSets` . The following code snippet shows how the new `ResultSet` is created. There can be only one `ResultSet` at a time. (Database/DriverSkeleton/SDriver.cxx)

```
Reference< XResultSet > SAL_CALL OStatement_Base::executeQuery( const ::rtl::OUString& sql )
                throw(SQLException, RuntimeException)
{
        ::osl::MutexGuard aGuard( m_aMutex );
        checkDisposed(OStatement_BASE::rBHelper.bDisposed);


        Reference< XResultSet > xRS = NULL;
        // create a resultset as result of executing the sql statement
        // something needs to be done here :-)
        m_xResultSet = xRS; // we nedd a reference to it for later use
        return xRS;
}
```

The `executeUpdate()` methods only return the rows that were affected by the given SQL statement. The last method execute returns true when a `ResultSet` object is returned when calling the method `getResultSet()`, otherwise it returns false. All other methods have to be implemented.

PreparedStatement

The `PreparedStatement` is used when an SQL statement should be executed more than once. In addition to the statement class, it must support the ability to provide information about the parameters when they exist. For this reason, this class must support the `com.sun.star.sdbc.XResultSetMetaDataSupplier` interface and also the `com.sun.star.sdbc.XParameters` interface to set values for their parameters.

Result Set

The `ResultSet` needs to be implemented. For the first step, only forward `ResultSet`s could be implemented, but it is recommended to support all `ResultSet` methods.

12.6.6 Support Scalar Functions

SDBC supports numeric, string, time, date, system, and conversion functions on scalar values. The Open Group CLI specification provides additional information on the semantics of the scalar functions. The functions supported are listed below for reference.

If a DBMS supports a scalar function, the driver should also. Scalar functions are supported by different DBMSs with different syntax, it is the driver's job to map the functions into the appropriate syntax or to implement the functions directly in the driver.

By calling metadata methods, a user can find out which functions are supported. For example, the method `XdatabaseMetaData.getNumericFunctions()` returns a comma separated list of the Open Group CLI names of the numeric functions supported. Similarly, the method `XDatabaseMetaData.getStringFunctions()` returns a list of string functions supported.

In the following table, the scalar functions are listed by category.

Open Group CLI Numeric Functions

Numeric Function	**Function Returns**
`ABS(number)`	Absolute value of number
`ACOS(float)`	Arccosine, in radians, of float
`ASIN(float)`	Arcsine, in radians, of float
`ATAN(float)`	Arctangent, in radians, of float
`ATAN2(float1, float2)`	Arctangent, in radians, of float2 / float1
`CEILING(number)`	Smallest integer >= number
`COS(float)`	Cosine of float radians
`COT(float)`	Cotangent of float radians
`DEGREES(number)`	Degrees in number radians
`EXP(float)`	Exponential function of float
`FLOOR(number)`	Largest integer <= number
`LOG(float)`	Base e logarithm of float
`LOG10(float)`	Base 10 logarithm of float

Numeric Function	Function Returns
MOD(integer1, integer2)	Remainder for integer1 / integer2
PI()	The constant pi
POWER(number, power)	number raised to (integer) power
RADIANS(number)	Radians in number degrees
RAND(integer)	Random floating point for seed integer
ROUND(number, places)	number rounded to places places
SIGN(number)	-1 to indicate number is < 0; 0 to indicate number is = 0; 1 to indicate number is > 0
SIN(float)	Sine of float radians
SQRT(float)	Square root of float
TAN(float)	Tangent of float radians
TRUNCATE(number, places)	number truncated to places places

Open Group CLI String Functions

String Functions	Function Returns
ASCII(string)	Integer representing the ASCII code value of the leftmost character in string.
CHAR(code)	Character with ASCII code value code, where the code is between 0 and 255.
CONCAT(string1, string2)	Character string formed by appending string2 to string1. If a string is null, the result is DBMS-dependent.
DIFFERENCE(string1, string2)	Integer indicating the difference between the values returned by the function SOUNDEX for string1 and string2.
INSERT(string1, start, length, string2)	A character string formed by deleting length characters from string1 beginning at the start, and inserting string2 into string1 at the start.
LCASE(string)	Converts all uppercase characters in string to lowercase.
LEFT(string, count)	The count leftmost characters from string.
LENGTH(string)	Number of characters in string, excluding trailing blanks.
LOCATE(string1, string2 [, start])	Position in string2 of the first occurrence of string1, searching from the beginning of string2. If start is specified, the search begins from position start. A 0 is returned if string2 does not contain string1. Position 1 is the first character in string2.
LTRIM(string)	Characters of string with leading blank spaces removed.
REPEAT(string, count)	A character string formed by repeating string count times.
REPLACE(string1, string2, string3)	Replaces all occurrences of string2 in string1 with string3.
RIGHT(string, count)	The count rightmost characters in string.
RTRIM(string)	The characters of string with no trailing blanks.
SOUNDEX(string)	A character string that is data source-dependent, representing the sound of the words in string, such as a four-digit SOUNDEX code, or a phonetic representation of each word.
SPACE(count)	A character string consisting of count spaces.

String Functions	Function Returns
SUBSTRING(string, start, length)	A character string formed by extracting length characters from string beginning at start.
UCASE(string)	Converts all lowercase characters in string to uppercase.

Open Group CLI Time and Date Functions

Time and Date Functions	Function Returns
CURDATE()	The current date as a date value.
CURTIME()	The current local time as a time value.
DAYNAME(date)	A character string representing the day component of the date. The name for the day is specific to the data source.
DAYOFMONTH(date)	An integer from 1 to 31 representing the day of the month in date.
DAYOFWEEK(date)	An integer from 1 to 7 representing the day of the week in date, where 1 represents Sunday.
DAYOFYEAR(date)	An integer from 1 to 366 representing the day of the year in date.
HOUR(time)	An integer from 0 to 23 representing the hour component of time.
MINUTE(time)	An integer from 0 to 59 representing the minute component of time.
MONTH(date)	An integer from 1 to 12 representing the month component of date.
MONTHNAME(date)	A character string representing the month component of date. The name for the month is specific to the data source.
NOW()	A timestamp value representing the current date and time.
QUARTER(date)	An integer from 1 to 4 representing the quarter in date, where 1 represents January 1 through March 31.
SECOND(time)	An integer from 0 to 59 representing the second component of time.
TIMESTAMPADD(interval, count, timestamp)	A timestamp calculated by adding count interval(s) to timestamp. Interval may be one of the following: SQL_TSI_FRAC_SECOND, SQL_TSI_SECOND, SQL_TSI_MINUTE, SQL_TSI_HOUR, SQL_TSI_DAY, SQL_TSI_WEEK, SQL_TSI_MONTH, SQL_TSI_QUARTER, or SQL_TSI_YEAR.
TIMESTAMPDIFF(interval, timestamp1, timestamp2)	An integer representing the number of interval(s) by which timestamp2 is greater than timestamp1. Interval may be one of the following:SQL_TSI_FRAC_SECOND, SQL_TSI_SECOND, SQL_TSI_MINUTE, SQL_TSI_HOUR, SQL_TSI_DAY, SQL_TSI_WEEK, SQL_TSI_MONTH, SQL_TSI_QUARTER, or SQL_TSI_YEAR
WEEK(date)	An integer from 1 to 53 representing the week of the year in date.
YEAR(date)	An integer representing the year component of date.

Open Group CLI System Functions

System Functions	Function Returns
DATABASE()	Name of the database.
IFNULL(expression, value)	Value if the expression is null; expression if expression is not null.

System Functions	Function Returns
`USER()`	User name in the DBMS.

Open Group CLI Conversion Functions

Conversion Function	Function Returns
`CONVERT(value, SQLtype)`	Value converted to SQLtype where SQLtype may be one of the following SQL types: BIGINT, BINARY, BIT, CHAR, DATE, DECIMAL, DOUBLE, FLOAT, INTEGER, LONGVARBINARY, LONGVARCHAR, REAL, SMALLINT, TIME, TIMESTAMP, TINYINT, VARBINARY, or VARCHAR.

Handling Unsupported Functionality

Some variation is allowed for drivers written for databases that do not support certain functionality. For example, some databases do not support out parameters with stored procedures. In this case, the `CallableStatement` methods that deal with out parameters (`registerOutParameter` and the various `XCallableStatement.getXXX()` methods) do not apply, and they should be implemented in such a way that they throw a `com.sun.star.sdbc.SQLException`.

The following features are optional in drivers for DBMSs that do not support them. When a DBMS does not support a feature, the methods that support the feature may throw a `SQLException`. The following list of optional features indicate if the `com.sun.star.sdbc.XDatabaseMetaData` methods are supported by the DBMS and driver.

- scrollable result sets: `supportsResultSetType()`
- modifiable result sets: `supportsResultSetConcurrency()`
- batch updates: `supportsBatchUpdates()`
- SQL3 data types: `getTypeInfo()`
- storage and retrieval of Java objects:
  - `getUDTs()` returns descriptions of the user defined types in a given schema
  - `getTypeInfo()` returns descriptions of the data types available in the DBMS.

13 Forms

13.1 Introduction

Forms offer a method of control-based data input. A form or *form document* consists of a set of controls, where each one enters a single piece of data. In a simple case, this could be a plain text field allowing you to insert some text without any word breaks. When we speak of forms, we mean forms and controls, because these cannot be divided.

If an internet site asks you for information, for example, for a product registration you are presented with fields to enter your name, your address and other information. These are HTML forms.

Basically, this is what OpenOffice.org forms do. They enhance nearly every document with controls for data input. This additional functionality put into a document is called the *form layer* within the scope of this chapter.

The most basic functionality provides the controls for HTML form documents mentioned above: If you open an HTML document with form elements in OpenOffice.org Writer, these elements are represented by components from `com.sun.star.form`.

The more enhanced functionality provides support for data-aware forms. These are forms and controls that are bound to a data source registered in OpenOffice.org to enter data into tables of a database. For more information about data sources and data access in general, refer to the *12 Database Access*.

When discussing forms, the difference between *form documents* and *logical forms* have to be distnguished. The form document refers to a document as a whole, and logical forms is a logical concept, basically a set of controls with additional properties. See below for details.
Within the scope of this chapter, when a "form" is referred to, we mean the logical form. The logical form is more interesting from the API programmer's perspective.

13.2 Models and Views

13.2.1 The Model-View Paradigm

A basic concept to understand about forms and controls in OpenOffice.org is the model-view paradigm. For a given element in your document,for example, a text field in your HTML form, it says that you have *exactly one* model and an arbitrary number of views.

The model is what is part of your document in that it describes how this element looks , and how it behaves. The model even exists when you do not have an open instance of your document. If it is stored in a file, the file contains a description of the model of your element.



In UNO, the simplest conceivable model is a component implementing `com.sun.star.beans.XPropertySet` only. Every aspect of the view could then be described by a single property. In fact, as you will see later, models for form controls are basically property sets.

The view is a visual representation of your model. It is the component which looks and behaves according to the requirements of the model. You can have multiple views for one model, and they would all look alike as the model describes it. The view is visible to the user. It is for visualizing the model and handles interactions with the user. The model, however, is merely a "dumb" container of data.

A good example to illustrate this is available in OpenOffice.org. Open an arbitrary document and choose the menu item **Window - New Window**. A second window is opened showing the same document displayed in the first window. This does not mean that the document was opened twice, it means you opened a second view of the same document, which is a difference. In particular, if you type some text in *one* of the windows, this change is visible in *both* windows. That is what the model-view paradigm is about: Keep your document data once in the model, and when you need to visualize the data to the user, or need interaction from the user that modifies the document, create views to the model as needed.

Between model and view a 1:n relationship exists:



Illustration 193



Note that the relation is directed. Usually, a view knows its model, but the model itself does not know about the views which visualize it.

13.2.2 Models and Views for Form Controls

Form controls follow the model-view paradigm. This means if you have a form document that contains a control, there is a model describing the control's behavior and appearance, and a view that is the component the user is sees.



Note that the term "control" is ambiguous here. Usually, from the user's perspective, it is what is seen in the document. As the model-view paradigm may not be obvious to the user, the user tends to consider the visible representation and the underlying model of the control as one thing, that is, a user who refers to the control usually means the combination of the view and the model.
As opposed to the user's perspective, when the UNO API for the form layer refers to a control, this means the *view* of a form element, if not stated otherwise.

The base for the controls and models used in the form layer are found in the module `com.sun.star.awt`, the `com.sun.star.awt.UnoControl` and `com.sun.star.awt.UnoControlModel` services. As discussed later, the model hierarchy in `com.sun.star.form.component` extends the hierarchy of `com.sun.star.awt`, whereas the control hierarchy in `com.sun.star.form.control` is small.

Everything the model-view interaction for form controls is true for other UNO controls and UNO control models, as well. Another example for components that use the model-view paradigm are

the controls and control models in OpenOffice.org Basic dialogs (*11.5.2 Basic and Dialogs - Programming Dialogs and Dialog Controls - Dialog Controls*).

13.2.3 Model-View Interaction

When a model and a view interoperate, a data transfer in both directions is required, from the model to the view and conversely.

Consider a simple text field. The model for a control implements a `com.sun.star.form.component.TextField` service. This means it has a property `Text`, containing the current content of the field, and a property `BackgroundColor` specifying the color that should be used as background when drawing the text of the control.

First, if f the value of the `BackgroundColor` property is changed, the control is notified of the change. This is done by UNO listener mechanisms, such as the `com.sun.star.beans.XPropertyChangeListener` allowing the control to listen for changes to model properties and react accordingly. Here the control would have to redraw itself using the new background color.

In fact this is a common mechanism for the communication between model and view: The view adds itself as listener for any aspect of the model which could affect it, and when it is notified of changes, it adjusts itself to the new model state. This means that the model is always the passive part. The model does not know its views, or at least not as views, but only their role as listeners, while the views know their model.

On the other hand, if the view is used for interaction with the user, of the data needs to be propagated from the view to the model. The user enters data in a text field, and the change is reflected in the model. Remember that the user sees the *control* only, and everything affects the *control* in the first step. If the user interacts with the view with the intention of modifying the model, the view propagates changes to the model.

In our example, the user enters text into the control, the control *automatically* updates the respective property at the model (`Text`), thus modifying the document containing the model.

13.2.4 Form Layer Views

View Modes

An important aspect to know when dealing with forms is that the view for a form layer is in different modes. More precise, there is a *design mode* available, opposite to a *live mode*. In design mode, you design your form. interactively with OpenOffice.org by inserting new controls, resizing them, and modifying their properties,together with control models and shapes. although OpenOffice.org hides this. In live mode, the controls interact with the user for data input.

The live mode is the natural mode for forms views, because usually a form is designed once and used again.

The following example switches a given document view between the two modes: (Forms/DocumentViewHelper.java)

```
/** toggles the design mode of the form layer of active view of our sample document
*/
protected void toggleFormDesignMode() throws java.lang.Exception {
    // get a dispatcher for the toggle URL
    URL[] aToggleURL = new URL[] {new URL()};
```

```
    aToggleURL[0].Complete = new String(".uno:SwitchControlDesignMode");
    XDispatch xDispatcher = getDispatcher(aToggleURL);

    // dispatch the URL - this will result in toggling the mode
    PropertyValue[] aDummyArgs = new PropertyValue[] {};
    xDispatcher.dispatch(aToggleURL[0], aDummyArgs);
}
```

The basic idea is to dispatch the URL ".uno:SwitchControlDesignMode" into the current view. This triggers the same functionality as if the button **Design Mode On/Off** was pressed in OpenOffice.org.In fact, `SwitchControlDesignMode` is the UNO name for the slot triggered by this button.

Locating Controls

A common task when working with form documents using the OpenOffice.org API is to obtain controls. Given that there is a control model, and a view to the document it belongs to, you may want to know the control that is used to represent the model in that view. This is what the interface `com.sun.star.view.XControlAccess` at the controller of a document view is made for. (Forms/DocumentViewHelper.java)

```
/** retrieves a control within the current view of a document
    @param xModel
        specifies the control model which's control should be located
    @return
        the control tied to the model
*/
public XControl getControl(XControlModel xModel) throws com.sun.star.uno.Exception {
    XControlAccess xCtrlAcc = (XControlAccess)UnoRuntime.queryInterface(
        XControlAccess.class , m_xController);
    // delegate the task of looking for the control
    return xCtrlAcc.getControl(xModel);
}
```

Focussing Controls

To focus a specific control in your document, or more precisely, in one of the views of your document: (Forms/DocumentViewHelper.java)

```
/** sets the focus to a specific control
    @param xModel
        a control model. The focus is set to that control which is part of our view
        and associated with the given model.
*/
public void grabControlFocus(Object xModel) throws com.sun.star.uno.Exception {
    // look for the control from the current view which belongs to the model
    XControl xControl = getControl(xModel);

    // the focus can be set to an XWindow only
    XWindow xControlWindow = (XWindow)UnoRuntime.queryInterface(Xwindow.class, xControl);

    // grab the focus
    xControlWindow.setFocus();
}
```

As you can see, focussing controls is reduced to locating controls. Once you have located the control, the `com.sun.star.awt.XWindow` interface provides everything needed for focussing.

13.3 Form Elements in the Document Model

The model of a document is the data that is made persistent, so that all form elements are a part of it. Refer to chapter *6.1.1 Office Development - OpenOffice.org Application Environment - Overview - Framework API - Frame-Controller-Model Paradigm* for additional information. This is true for logical

forms, as well as for control models. Controls , that is, the view part of form elements, are not made persistent, thus are not accessible in the document model.

13.3.1 A Hierarchy of Models

The components in the form layer are organized hierarchically in an object tree. Their relationship is organized using the standard interfaces, such as `com.sun.star.container.XChild` and `com.sun.star.container.XIndexAccess`.

As in every tree, there is a root with inner nodes and leaves. There are different components described below that take on one or several of these roles.

FormComponent Service

The basis for all form related models is the `com.sun.star.form.FormComponent` service. Its basic characteristics are:

- it exports the `com.sun.star.container.XChild` interface
- it has a property `Name`
- it exports the `com.sun.star.lang.XComponent` interface

Form components have a parent and a name, and support lifetime control that the common denominator for form elements and logical forms, as well as for control models.

FormComponents Service

In the level above, a single form component is a container for components. Stepping away from the document model, you are looking for a specific form component, such as the model of a control, you pass where all the control models are attached. This is the `com.sun.star.form.FormComponents` component. The service offers basic container functionality, namely an access to its elements by index or by name), and a possibility to enumerate its elements.

Provided that you have a container at hand, the access to its elements is straightforward. For example, assume you want to enumerate all the elements in the container, and apply a specific action for every element. The `enumFormComponents()` method below does this by recursively enumerating the elements in a `com.sun.star.form.FormComponents` container. (Forms/FormLayer.java)

```
/** enumerates and prints all the elements in the given container
*/
public static void enumFormComponents(XNameAccess xContainer, String sPrefix)
       throws java.lang.Exception {
    // loop through all the element names
    String aNames[] = xContainer.getElementNames();
    for (int i=0; i<aNames.length; ++i) {
        // print the child name
        System.out.println(sPrefix + aNames[i]);

        // check if it's a FormComponents component itself
        XServiceInfo xSI = (XServiceInfo)UnoRuntime.queryInterface(XServiceInfo.class,
            xContainer.getByName(aNames[i]));

        if (xSI.supportsService("com.sun.star.form.FormComponents")) {
            // yep, it is
            // -> step down
            XNameAccess xChildContainer = (XnameAccess)UnoRuntime.queryInterface(
                XNameAccess.class, xSI);
            enumFormComponents(xChildContainer, new String(" ") + sPrefix);
```

```
        }
    }
}

/** enumerates and prints all the elements in the given container, together with the container itself
*/
public static void enumFormComponents(XNameAccess xContainer) throws java.lang.Exception {
    XNamed xNameAcc = (XNamed)UnoRuntime.queryInterface(XNamed.class, xContainer);
    String sObjectName = xNameAcc.getName();
    System.out.println( new String("enumerating the container named \"") + sObjectName +
        new String("\"\n"));

    System.out.println(sObjectName);
    enumFormComponents(xContainer, " ");
}
```

Logical Forms

Forms as technical objects are also part of the document model. In contrast to control models, forms do not have a view representation. For every control model, there is a control the user interacts with, and presents the data back to the user. For the form, there is no view component.

The basic service for logical forms is `com.sun.star.form.component.Form`. See below for details regarding this service. For now, we are interested in that it exposes the `com.sun.star.form.FormComponent` service, as well as the `com.sun.star.form.FormComponents` service. This means it is part of a form component container, and it is a container. Thus, in our hierarchy of models, it can be any node, such as an inner node having children, that is, other form components,, as well as a leaf node having no children, but a parent container. **Of course b**oth of these roles are not exclusive. This is how data aware forms implement master-detail relationships. Refer to the *13.5 Forms - Data Awareness.*

Forms Container

In our model hierarchy, we have inner nodes called the logical forms, and the basic element called the form component. As in every tree, our hierarchy has a root, that is, an instance of the `com.sun.star.form.Forms` service. This is nothing more than an instance of `com.sun.star.form.FormComponents`. In fact, the differentiation exists for a non-ambiguous runtime instantiation of a root.



Note that the `com.sun.star.form.Forms` service does not state that components implementing it are a `com.sun.star.form.FormComponent`. This means this service acts as a tree root only, opposite to a `com.sun.star.form.Forms` that is a container, as well as an element, thus it can be placed anywhere in the tree.

Actually, it is not necessary for external components to instantiate a service directly. Every document has at least one instance of it. A root forms container is tied to a draw page, which is an element of the document model, as well. Refer to `com.sun.star.drawing.DrawPage`. A page optionally supports the interface `com.sun.star.form.XFormsSupplier` giving access to the collection. In the current OpenOffice.org implementation, Writer and Calc documents fully support draw pages supplying forms.

The following example shows how to obtain a root forms collection, if the document model is known which is denoted with `s_aDocument`. (Forms/DocumentHelper.java)

```
/** gets the <type scope="com.sun.star.drawing">DrawPage</type> of our sample document
*/
public static XDrawPage getDocumentDrawPage() throws java.lang.Exception {
    XDrawPage xReturn;

    // in case of a Writer document, this is rather easy: simply ask the XDrawPageSupplier
    XDrawPageSupplier xSuppPage = (XDrawPageSupplier)UnoRuntime.queryInterface(
        XDrawPageSupplier.class, s_aDocument);
    xReturn = xSuppPage.getDrawPage();
```

```
    if (null == xReturn) {
        // the model itself is no draw page supplier - then it may be an Impress or Calc
        // (or any other multi-page) document
        XDrawPagesSupplier xSuppPages = (XDrawPagesSupplier)UnoRuntime.queryInterface(
            XDrawPagesSupplier.class, s_aDocument);
        XDrawPages xPages = xSuppPages.getDrawPages();

        xReturn = (XdrawPage)UnoRuntime.queryInterface(XDrawPage.class, xPages.getByIndex(0));

        // Note that this is not really error-proof code: If the document model does not support the
        // XDrawPagesSupplier interface, or if the pages collection returned is empty, this will break.
    }

    return xReturn;
}

/** retrieves the root of the hierarchy of form components
*/
public static XNameContainer getFormComponentTreeRoot() throws java.lang.Exception {
    XFormsSupplier xSuppForms = (XFormsSupplier)UnoRuntime.queryInterface(
        XFormsSupplier.class, getDocumentDrawPage());

    XNameContainer xFormsCollection = null;
    if (null != xSuppForms) {
        xFormsCollection = xSuppForms.getForms();
    }
    return xFormsCollection;
}
```

Form Control Models

The control models are discussed in these sections. The basic service for a form layer control model is `com.sun.star.form.FormControlModel` that is discussedin more detail below. A form control model promises to support the `com.sun.star.form.FormComponent` service, meaning that it can act as a child in our model hierarchy.

In addition, it does not claim that the `com.sun.star.form.FormComponents` service (plural s) is supported meaning that form control models are leaves in our object tree. The only exception from this is the grid control model. It is allowed to have children representing the models of the columns.

An overview of the whole model tree has been provided. With the code fragments introduced above, the following code dumps a model tree to the console:

```
    // dump the form component tree
    enumFormComponents(getFormComponentTreeRoot());
```

13.3.2 Control Models and Shapes

There is more to know about form components in a document.

From *9.3.2 Drawing - Working with Drawing Documents - Shapes*, you already know about shapes. They are also part of a document model. The control shapes, `com.sun.star.drawing.ControlShape` are made to be tied to control models. They are specialized to fully integrate form control models into a document.

In theory, there can be a control shape without a model tied to it, or a control model which is part of the form component hierarchy, but not associated with any shape. In the first case, an empty shape is displayed in the document view. In the second case, you see nothing. It is possible to have a shape which is properly tied to a control model, but the control model is not part of the form component hierarchy. The model can not interact with the rest of the form layer. For example, it is unable to take advantage of its data awareness capabilities.

The user interface of OpenOffice.org does not allow the creation of orphaned objects, but you can create them using the API. When dealing with controls through the API, ensure that there is always a valid relationship between forms, control models, and shapes.

A complete object structure in a document model with respect to the components relevant for our form layer looks the following:

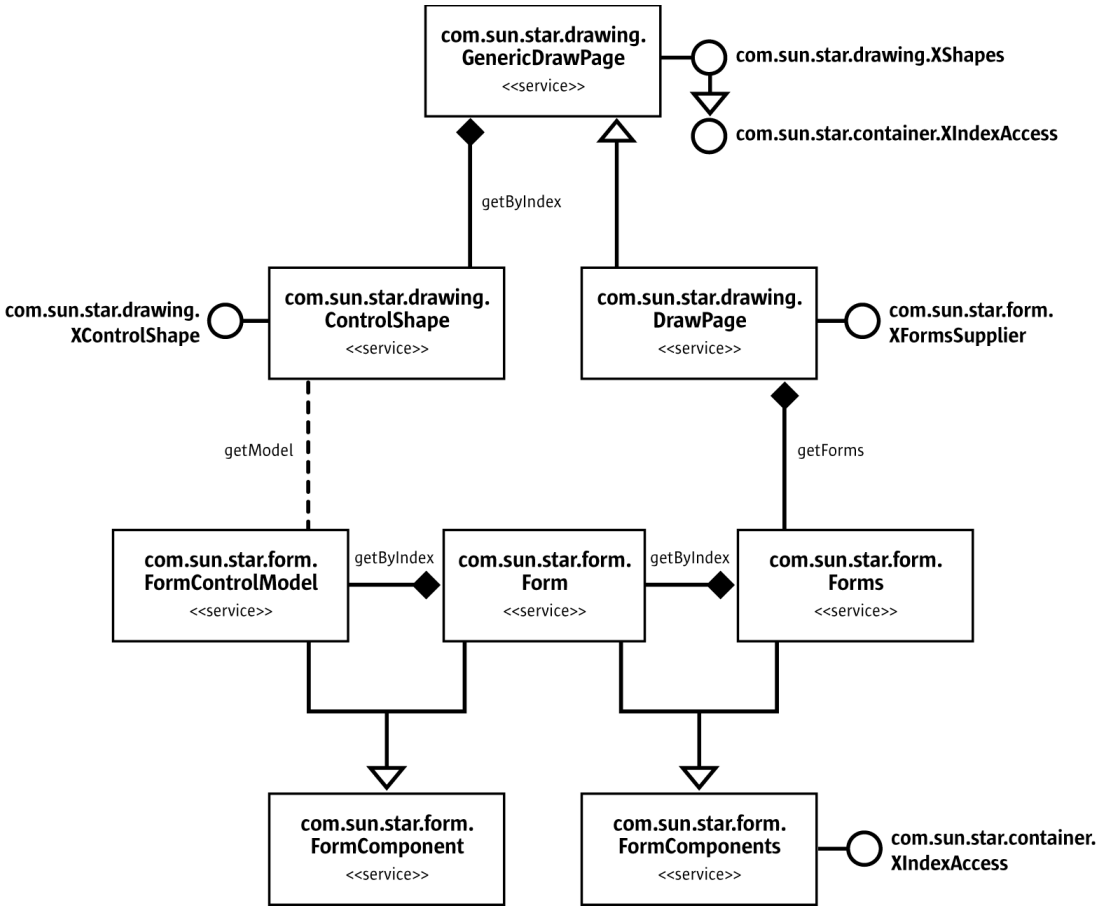


Illustration 194

Programmatic Creation of Controls

As a consequence from the previous paragraph, we now know that to insert a form control, we need to insert a control shape and control model into the document's model.

The following code fragment accomplishes that: (Forms/FormLayer.java)

```
/** creates a control in the document

    <p>Note that <em>control<em> here is an incorrect terminology. What the method really does is
    it creates a control shape, together with a control model, and inserts them into the document model.
    This will result in every view to this document creating a control described by the model-shape
    pair.</p>

    @param sFormComponentService
        the service name of the form component to create, e.g. "TextField"
    @param nXPos
        the abscissa of the position of the newly inserted shape
    @param nXPos
        the ordinate of the position of the newly inserted shape
    @param nWidth
        the width of the newly inserted shape
    @param nHeight
```

```
        the height of the newly inserted shape
    @return
        the property access to the control's model
*/
public static XPropertySet createControlAndShape(String sFormComponentService, int nXPos,
        int nYPos, int nWidth, int nHeight) throws java.lang.Exception {
    // let the document create a shape
    XMultiServiceFactory xDocAsFactory = (XMultiServiceFactory)UnoRuntime.queryInterface(
        XMultiServiceFactory.class, s_aDocument);
    XControlShape xShape = (XControlShape)UnoRuntime.queryInterface(XControlShape.class,
        xDocAsFactory.createInstance("com.sun.star.drawing.ControlShape"));

    // position and size of the shape
    xShape.setSize(new Size(nWidth * 100, nHeight * 100));
    xShape.setPosition(new Point(nXPos * 100, nYPos * 100));

    // and in a OOo Writer doc, the anchor can be adjusted
    XPropertySet xShapeProps = (XPropertySet)UnoRuntime.queryInterface(XPropertySet.class, xShape);
    TextContentAnchorType eAnchorType = TextContentAnchorType.AT_PAGE;
    if (classifyDocument(s_aDocument) == DocumentType.WRITER) {
        eAnchorType = TextContentAnchorType.AT_PARAGRAPH;
    }
    xShapeProps.setPropertyValue("AnchorType", eAnchorType);

    // create the form component (the model of a form control)
    String sQualifiedComponentName = "com.sun.star.form.component." + sFormComponentService;
    XControlModel xModel = (XControlModel)UnoRuntime.queryInterface(XControlModel.class,
        s_aMSF.createInstance(sQualifiedComponentName));

    // knitt them
    xShape.setControl(xModel);

    // add the shape to the shapes collection of the document
    XShapes xDocShapes = (XShapes)UnoRuntime.queryInterface(XShapes.class, getDocumentDrawPage());
    xDocShapes.add(xShape);

    // and outta here with the XPropertySet interface of the model
    XPropertySet xModelProps = (XpropertySet)UnoRuntime.queryInterface(
        XpropertySet.class, xModel);
    return xModelProps;
}
```

Looking at the example above, the basic procedure is:

- create and initialize a shape
- create a control model
- announce the control model to the shape
- insert the shape into the shapes collection of a draw page

The above does not mention about inserting the control model into the form component hierarchy, which is a contradiction of our previous discussion. We have previously said that every control model must be part of this hierarchy to prevent corrupted documents, but it is not harmful.

In every document, when a new control shape is inserted into the document, through the API or an interaction with a document's view, the control model is checked if it is a member of the model hierarchy. If it is not, it is *automatically* inserted. Moreover, if the hierarchy does not exist or is incomplete, for example, if the draw page does not have a forms collection, or this collection does not contain a form, this is also corrected automatically.

With the code fragment above applied to a new document, a logical form is created automatically, inserted into the forms hierarchy, and the control model is inserted into this form.



Note that this is an implementation detail. Internally, there is an instance listening at the page's shapes, that reacts upon insertions. In theory, there could be other implementations of OpenOffice.org API that do not contain this mechanism. In practice, the only known implementation is OpenOffice.org.



Note that the order of operations is important. If you insert the shape into the page's shape collection, and tie it to its control model after, the document would be corrupted: Nobody would know about this new model then, and it would not be inserted properly into the form component hierarchy, unless you do this.

You may have noticed that there is nothing about the view. We only created a control model. As you can see in the complete example for this chapter, when you have an open document, and insert a model and a shape, a control (the visual representation) is also created or else you would not see anything that looks like a control.

The control and model have a model-view relationship. If the document window is open, this window is the document *view*. If the document or the *model* is modified by inserting a control model, the view for every open view for this document reacts appropriately and creates a control as described by the model. The `com.sun.star.awt.UnoControlModel:DefaultControl` property describes the service to be instantiated when automatically creating a control for a model.

13.4 Form Components

13.4.1 Basics

According to the different *form document* types, there are different components in the `com.sun.star.form` module serving different purposes. Basically, we distinguish between *HTML form functionality* and *data awareness functionality* that are covered by the form layer API.

Control Models

As you know from *13.3.1 Forms - Form Elements in the Document Model - Hierarchy - Form Control Models*, the base for all our control models is the `com.sun.star.form.FormControlModel` service. Let us look at the most relevant elements of the declaration of this service and what a component must do to support it:

`com.sun.star.awt.UnoControlModel`
: This service specifies that a form control model complies to everything required for a control model by the UNO windowing toolkit as described in module `com.sun.star.awt`. This means support for the `com.sun.star.awt.XControlModel` interface, for property access and persistence.

`com.sun.star.form.FormComponent`
: This service requires a form control model is part of a form component hierarchy. Refer to chapter *13.3.1 Forms - Form Elements in the Document Model - Hierarchy.*

`com.sun.star.beans.XPropertyState`
: This optional interface allows the control model properties to have a *default value*. All known implementations of the `FormControlModel` service support this interface.

`com.sun.star.form.FormControlModel:ClassId`
: This property determines the class of a control model you have , and it assumes a value from the `com.sun.star.form.FormComponentType` enumeration. The same is done using the `com.sun.star.lang.XServiceInfo` interface that is supported by every component, and as shown below it can be indispensable. Using the `com.sun.star.form.FormControlModel:ClassId` property is faster.



Note that the `com.sun.star.form.FormControlModel` service does not state anything about data awareness. It describes the requirements for a control model which can be part of a form layer.

See chapter *13.5 Forms - Data Awareness* for additional information about the controls which are data aware.

The following example shows how to determine the type of a control model using the `ClassId` property introduced above: (Forms/FLTools.java)

```
/** retrieves the type of a form component.
    <p>Speaking strictly, the function recognizes more than form components. Especially,
    it survives a null argument. which means it can be safely applied to the a top-level
    forms container; and it is able to classify grid columns (which are no form components)
    as well.</p>
*/
static public String classifyFormComponentType(XPropertySet xComponent)
        throws com.sun.star.uno.Exception {
    String sType = "<unknown component>";

    XServiceInfo xSI = (XserviceInfo)UnoRuntime.queryInterface(XServiceInfo.class, xComponent);

    XPropertySetInfo xPSI = null;
    if (null != xComponent)
        xPSI = xComponent.getPropertySetInfo();

    if ( ( null != xPSI ) && xPSI.hasPropertyByName("ClassId")) {
        // get the ClassId property
        XPropertySet xCompProps = (XPropertySet)UnoRuntime.queryInterface(
            XPropertySet.class, xComponent);

        Short nClassId = (Short)xCompProps.getPropertyValue("ClassId");
        switch (nClassId.intValue())
        {
            case FormComponentType.COMMANDBUTTON: sType = "Command button"; break;
            case FormComponentType.RADIOBUTTON  : sType = "Radio button"; break;
            case FormComponentType.IMAGEBUTTON  : sType = "Image button"; break;
            case FormComponentType.CHECKBOX     : sType = "Check Box"; break;
            case FormComponentType.LISTBOX      : sType = "List Box"; break;
            case FormComponentType.COMBOBOX     : sType = "Combo Box"; break;
            case FormComponentType.GROUPBOX     : sType = "Group Box"; break;
            case FormComponentType.FIXEDTEXT    : sType = "Fixed Text"; break;
            case FormComponentType.GRIDCONTROL  : sType = "Grid Control"; break;
            case FormComponentType.FILECONTROL  : sType = "File Control"; break;
            case FormComponentType.HIDDENCONTROL: sType = "Hidden Control"; break;
            case FormComponentType.IMAGECONTROL : sType = "Image Control"; break;
            case FormComponentType.DATEFIELD    : sType = "Date Field"; break;
            case FormComponentType.TIMEFIELD    : sType = "Time Field"; break;
            case FormComponentType.NUMERICFIELD : sType = "Numeric Field"; break;
            case FormComponentType.CURRENCYFIELD: sType = "Currency Field"; break;
            case FormComponentType.PATTERNFIELD : sType = "Pattern Field"; break;

            case FormComponentType.TEXTFIELD    :
                // there are two known services with this class id: the usual text field,
                // and the formatted field
                sType = "Text Field";
                if (( null != xSI) && xSI.supportsService(
                        "com.sun.star.form.component.FormattedField")) {
                    sType = "Formatted Field";
                }
                break;

            default:
                break;
        }
    }
    else {
        if ((null != xSI) && xSI.supportsService("com.sun.star.form.component.DataForm")) {
            sType = "Form";
        }
    }

    return sType;
}
```

Note the special handling for the value `com.sun.star.form.FormComponentType:TEXTFIELD`. There are two different services where a component implementing them is required to act as text field, the `com.sun.star.form.component.TextField` and `com.sun.star.form.component.FormattedField`. Both services describe a text component, thus both have a class id of `com.sun.star.form.FormComponentType:TEXTFIELD`. To distinguish between them, ask the components for more details using the `com.sun.star.lang.XServiceInfo` interface.

Forms

The OpenOffice.org API features different kinds of forms, namely the `com.sun.star.form.component.Form`, `com.sun.star.form.component.HTMLForm`, and `com.sun.star.form.component.DataForm`. The two different aspects described with these services are HTML forms used in HTML documents, and data aware forms used to access databases. Data awareness is discussed thoroughly in *13.5 Forms - Data Awareness.*



Though different services exist for HTML and data aware forms, there is only one form implementation in OpenOffice.org htat implements both services simultaneously.

The common denominator of HTML forms and data aware forms is described in the `com.sun.star.form.component.Form` service. It includes the `FormComponent` and `FormComponents` service, in addition to the following elements:

`com.sun.star.form.XForm`
: This interface identifies the component as a form that can be done with other methods, such as the `com.sun.star.lang.XServiceInfo` interface. The `com.sun.star.form.XForm` interface distinguishes a form component as a form. The `XForm` interface inherits from `com.sun.star.form.XFormComponent` to indicate the difference, and does not add any further operations.

`com.sun.star.awt.XTabControllerModel`
: This is used for controlling tab ordering and control grouping. As a logical form is a container for control models, it is a natural place to administer information about the relationship of its control children. The tab order, that is, the order in which the focus travels through the controls associated with the control models when the user presses the **Tab** key, is a relationship, and thus is maintained on the form.

  Note that changing the tab order through this interface also affects the models. The `com.sun.star.form.FormControlModel` service has an optional property `TabIndex`that contains the relative position of the control in the tabbing order. For example, a straightforward implementation of `com.sun.star.awt.XTabControllerModel:setControlModels()` would be simply to adjust all the `TabIndex` properties of the models passed to this method.

13.4.2 HTML Forms

The `com.sun.star.form.component.HTMLForm` service reflects the requirements for HTML form documents. Looking at HTML specifications, you can submit forms using different encodings and submit methods, and reset forms. The `HTMLForm` service description reflects this by supporting the interfaces `com.sun.star.form.XReset` and `com.sun.star.form.XSubmit`, as well as some additional properties related to the submit functionality.

The semantics of these interfaces and properties are straightforward.For additional details, refer to the service description, as well as the HTML specification.

13.5 Data Awareness

A major feature of forms in OpenOffice.org is that they can be data aware. You create form documents where the user manipulates data from a database that is accessible in OpenOffice.org. For more details about data sources, refer to chapter *12 Database Access.* This includes data from any table of a database, or data from a query based on one or more tables.

The basic idea is that a logical form cis associated with a database result set. A form control model, which is a child of that form, is bound to a field of this result set, exchanging the data entered by the user with the result set field.

13.5.1 Forms

Forms as Row Sets

Besides forms, there is already a component that supports a result set, the `com.sun.star.sdb.RowSet`.If you look at the `com.sun.star.form.component.DataForm`, a Data-Form also implements the `com.sun.star.sdb.RowSet` service, and extends it with additional functionality. Row sets are described in *12.3.1 Database Access - Manipulating Data - The RowSet Service*.

Loadable Forms

A major difference of data forms compared to the underlying row set is the that forms are *loaded*, and t provide an interface to manipulate this state.

```
XLoadable xLoad = (XLoadable)FLTools.getParent(aControlModel, XLoadable.class);
xLoad.reload();
```

Loading is the same as executing the underlying row set, that is, invoking the `com.sun.star.sdbc.XRowSet:execute()` method. The `com.sun.star.form.XLoadable` is designed to fit the needs of a form document, for example, it a unloads an already loaded form.

The example above shows how to reload a form. Reloading is executing the row set again. Using `reload` instead of `execute` has the advantage of advanced listener mechanisms:

Look at the `com.sun.star.form.XLoadable` interface. You can add a `com.sun.star.form.XLoadListener`. This listener not only tells you when load-related events have occurred that is achieved by the `com.sun.star.sdbc.XRowSetListener`, but also when they are *about* to happen. In a complex scenario where different listeners are added to different aspects of a form, you use the `com.sun.star.form.XLoadable:reloading()` call to disable all other listeners temporarily. Re-executing a row set is a complex process, thus it triggers a lot of events that are only an after effect of the re-execution.



Though all the functionality provided by `com.sun.star.form.XLoadable` can be simulated using the `com.sun.star.sdbc.XRowSet` interface, you should always use the former. Due to the above-mentioned, more sophisticated listener mechanisms, implementations have a chance to do loading, reloading and unloading much smoother then.

An additional difference between loading and executing is the positioning of the row set: When using `com.sun.star.sdbc.XRowSet:execute()`, the set is positioned *before* the first record. When you use `com.sun.star.form.XLoadable:load()`, the set is positioned *on* the first record, as you would expect from a form.

Sub Forms

A powerful feature of OpenOffice.org are sub forms. This does not mean that complete form documents are embedded into other form documents, instead sub form relationships are realized by nesting *logical* forms in the form component hierarchy.

When a form notices that its parent is not the forms container when it is loaded and in live mode, but is dependent on another form, it no longer acts as a top-level form. Whenever the parent or *master* form moves to another record, the content of the sub or *detail* form is re-fetched. This way, the content of the sub form is made dependent on the actual value of one or more fields of the parent form.

Typical use for a relationship are tables that are linked through key columns, usually in a 1:n relationship. You use a master form to travel trough all records of the table on the 1 side of the relationship, and a detail form that shows the records of the table on the n side of the relationship where the foreign key matches the primary key of the master table.

To create nested forms at runtime, use the following example: (Forms/FormLayer.java)

```
// retrieve or create the master form
m_xMasterForm = ....

// bind it to the salesman table
m_xMasterForm.setPropertyValue("DataSourceName", m_aParameters.sDataSourceName);
m_xMasterForm.setPropertyValue("CommandType", new Integer(CommandType.TABLE));
m_xMasterForm.setPropertyValue("Command", "SALESMAN");

// create the details form
XIndexContainer xSalesForm = m_aDocument.createSubForm(m_xMasterForm, "Sales");
XPropertySet xSalesFormProps = (XPropertySet)UnoRuntime.queryInterface(
    XPropertySet.class, xSalesForm);

// bind it to the all those sales belonging to a variable salesmen
xSalesFormProps.setPropertyValue("DataSourceName", m_aParameters.sDataSourceName);
xSalesFormProps.setPropertyValue("CommandType", new Integer( CommandType.COMMAND));
xSalesFormProps.setPropertyValue("Command",
    "SELECT * FROM SALES AS SALES WHERE SALES.SNR = :salesman");

// the master-details connection
String[] aMasterFields = new String[] {"SNR"};      // the field in the master form
String[] aDetailFields = new String[] {"salesman"}; // the name in the detail form
xSalesFormProps.setPropertyValue("MasterFields", aMasterFields);
xSalesFormProps.setPropertyValue("DetailFields", aDetailFields);
```

The code snippet works on the following table structure:

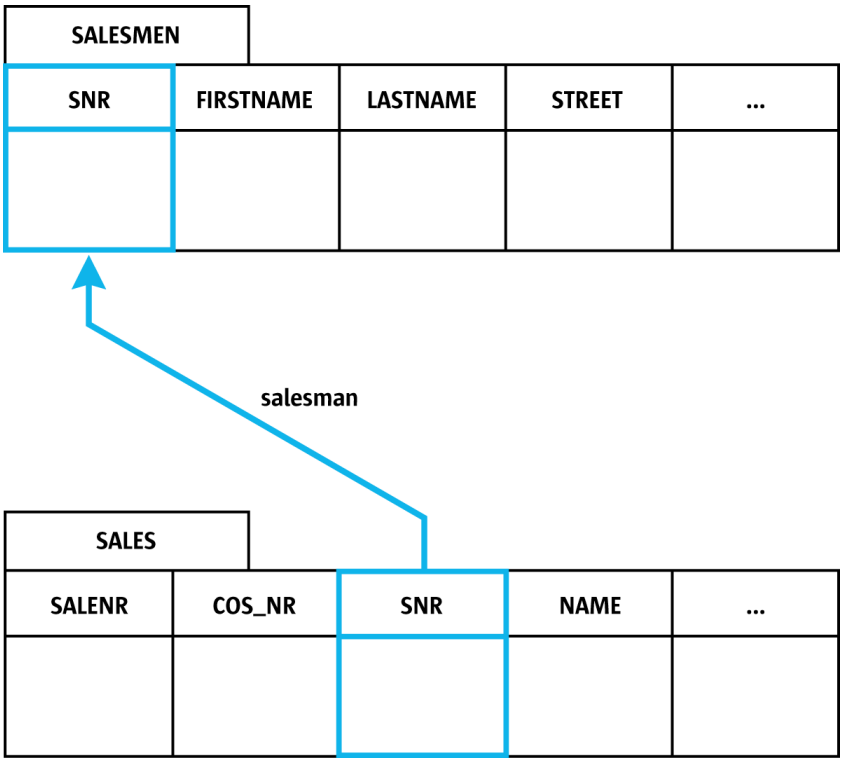


Illustration 195

The code is straigh forward, except setting up the connection between the two forms. The master form is bound to SALESMEN, and the detail form is bound to a statement that selects all fields from SALES, filtered for records where the foreign key, SALES.SNR, equals a parameter named `salesman`.

As soon as the `MasterFields` and `DetailFields` properties are set, the two forms are connected. Every time the cursor in the master form moves, the detail form reloads after filling the `salesman` parameter with the actual value of the master forms SNR column.

Filtering and Sorting

Forms support quick and easy filtering and sorting like the underlying row sets. For this, the properties `com.sun.star.sdb.RowSet:Filter`, `com.sun.star.sdb.RowSet:ApplyFilter` and `com.sun.star.sdb.RowSet:Order` area used. (Forms/SalesFilter.java)

```
// set this as filter on the form
String sCompleteFilter = "";
if ((null != sOdbcDate) && (0 != sOdbcDate.length())) {
    sCompleteFilter = "SALEDATE >= ";
    sCompleteFilter += sOdbcDate;
}
m_xSalesForm.setPropertyValue("Filter", sCompleteFilter);
m_xSalesForm.setPropertyValue("ApplyFilter", new Boolean(true));

// and reload the form
XLoadable xLoad = (XLoadable)UnoRuntime.queryInterface(XLoadable.class, m_xSalesForm);
xLoad.reload();
```

In this fragment, a filter string is built first. The `"SALEDATE >= {D '2002-12-02'}"` is an example for a filter string. In general, everything that appears after the `WHERE` clause of an SQL statement is set as a `Filter` property value. The same holds true for the `Order` property value and an `ORDER BY` clause.



Note the notation for the date in braces: This is the standard ODBC notation for date values, and it is the safest method to supply OpenOffice.org with date values. It also works if you are using non-ODBC data sources, as long as you do not switch on the **Native SQL** option. Refer to `com.sun.star.sdbc.Statement:EscapeProcessing`. OpenOffice.org understands and sometimes returns other notations, for instance, in the user interface where that makes sense, but these are locale-dependent, which means you have to know the current locale if you use them.

Then the `ApplyFilter` property is set to `true`. This is for safety, because the value of this property is unknown when creating a new form. Everytime you have a form or row set, and you want to change the filter, remember to set the `ApplyFilter` property at least once. Afterwards, `reload()` is called.

In general, `ApplyFilter` allows the user of a row set to enable or disable the current filter quickly without remembering it. To see what the effects of the current filter are, set `ApplyFilter` to `false` and reload the form.

Parameters

Data Aware Forms are based on statements. As with other topics in this chapter, this is not form specific, instead it is a functionality inherited from the underlying `com.sun.star.sdb.RowSet`. Statements contain parameters where some values are not specified, and are not dependent on actual values in the underlying tables. Instead they have to be filled each time the row set is executed, that is, the form is loaded or reloaded.

A typical example for a statement containing a parameter is

```
SELECT * FROM SALES WHERE SALES.SNR = :salesman
```

There is a named parameter `salesman`, which is filled before a row set based on a statement is executed. The orthodox method to use is the `com.sun.star.sdbc.XParameters` interface, exported by the row set.

However, forms allow another way. They export the `com.sun.star.form.XDatabaseParameterBroadcaster` interface that allows your component to add itself as a listener for an event which is triggered whenever the form needs parameter values.

In a form, filling parameters is a three-step procedure. Consider a form that needs three parameters for execution.

1. The master-detail relationship is evaluated. If the form's parent is a `com.sun.star.form.component.DataForm`, then the `MasterFields` and `DetailFields` properties are evaluated to fill in parameter values. For an example of how this relationship is evaluated, refer to chapter *13.5.1 Forms - Data Awareness - Forms - Sub Forms*.
2. If there are parameter values left, that is, not filled in, the calls to the `com.sun.star.sdbc.XParameters` interface are examined. All values previously set through this interface are filled in.
3. If there are still parameter values left, the `com.sun.star.form.XDatabaseParameterListeners` are invoked. Any component can add itself as a listener using the `com.sun.star.form.XDatabaseParameterBroadcaster` interface implemented by the form.
   The listeners then have the chance to fill in anything still missing.

Unfortunately, OpenOffice.org Basic scripts currently cannot follow the last step of this procedure —there is a known implementation issue which prevents this.

13.5.2 Data Aware Controls

The second part of the Data Awareness capabilities of OpenOffice.org are data aware controls. While a form is always associated with a complete result set, it *represents* this result set, a single control is bound to one data column that is part of the form which is the control's parent.

As always, the relevant information is stored in the control model. The basic service for control models which are data-aware is `com.sun.star.form.DataAwareControlModel`.

There are two connections between a control model and the column it is bound to:

DataField
: This is the property that determines the *name* of the field to bind to. Upon loading the form, a control model searches the data columns of the form for this name, and connects to it. An explanation for "connects" is provided below.
  Note that this property isa suggestion only. It tells the control model to connect to the data column, but this connection may fail for various reasons, for example, no such column may exist in the row set.
  Even if this property is set to a non-empty string, this does not mean anything about the control being connected.

BoundField
: Once a control model has connected itself to a data column, the respective column object is also remembered. This saves clients of a control model the effort to examine and handle the *DataField* , they simply rely on *BoundField*.
  Opposite to the *DataField* property, *BoundField* is reliable in that it is a valid column object if and only if the control is properly connected.

The overall relationship for data awareness is as follows:

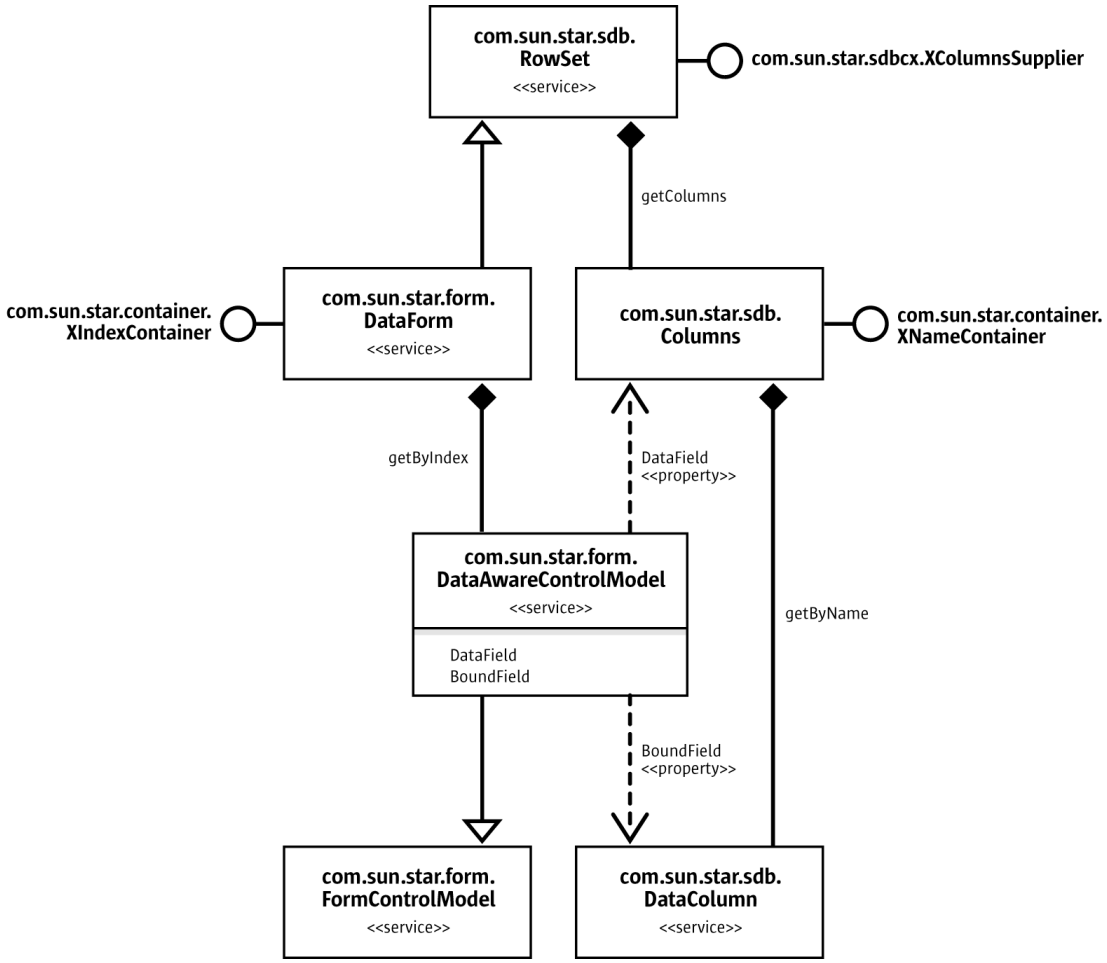


Illustration 196

Control Models as Bound Components

You expect that the control displays the current data of the column it is tied to. Current data means the data in the row that the `com.sun.star.form.component.DataForm` is currently located on. Now, the *control* does not know about data-awareness, only the control model does, but we already have a connection between the model and control: As described in the chapter about model-view interaction, *13.2.3 Forms - Models and Views - Model-View Interaction*, the control listens for changes to the model properties, as well as updates them when a user interacts with the control directly.

For instance, you know the `Text` property of a simple text input field, `com.sun.star.form.component.TextField`that is updated by the control when the user enters text. When the property is updated through any other means, the control reacts appropriately and adjusts the text it displays.

This mechanism is found in all controls. The only difference is the property used to determine the contents to be displayed. For instance, numeric controls `com.sun.star.form.component.NumericField` have a property `Value` representing the current numerical value to be displayed.
Although the name differs, all control models have a dedicated *content property*.

This is where the data-awareness comes in. A data-aware control model bound to a data column uses its content property to exchange data with this column. As soon as the column value changes, the model forwards the new value to its content property, and notifies its listeners. One of these listeners is the control that updates its display:

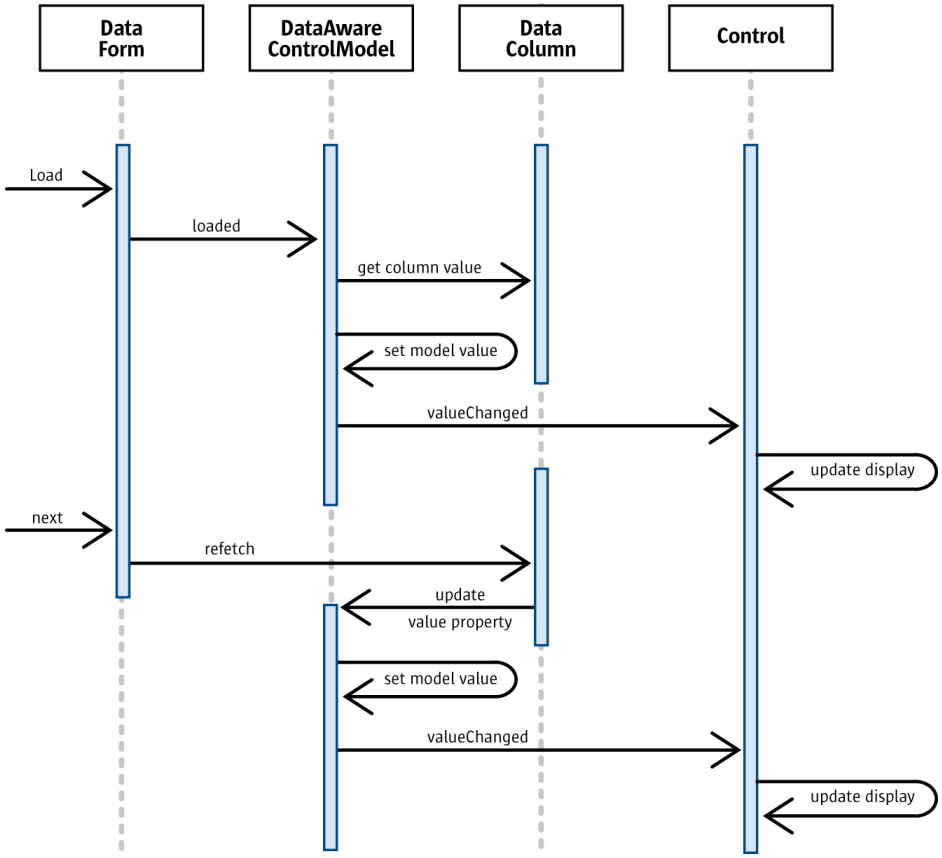


Illustration 197

Committing Controls

The second direction of the data transfer is back from what the user enters into the control. The text entered by a user is immediately forwarded to the value property of the control model. This way, both the control and the control model are always consistent.

Next, the content property is transferred into the data column the control is bound to. As opposed to the first step, this is not done automatically. Instead, this control is *committed* actively.

Committing is the process of transferring the current value of the control to the database column. The interface used for this is `com.sun.star.form.XBoundComponent` that provides the method `commit`. Note that the `XBoundComponent` is derived from `com.sun.star.form.XUpdateBroadcaster`. This means that listeners are added to a component to monitor and veto the committing of data.

The following diagram shows what happens when the user decides to save the current record after changing a control:

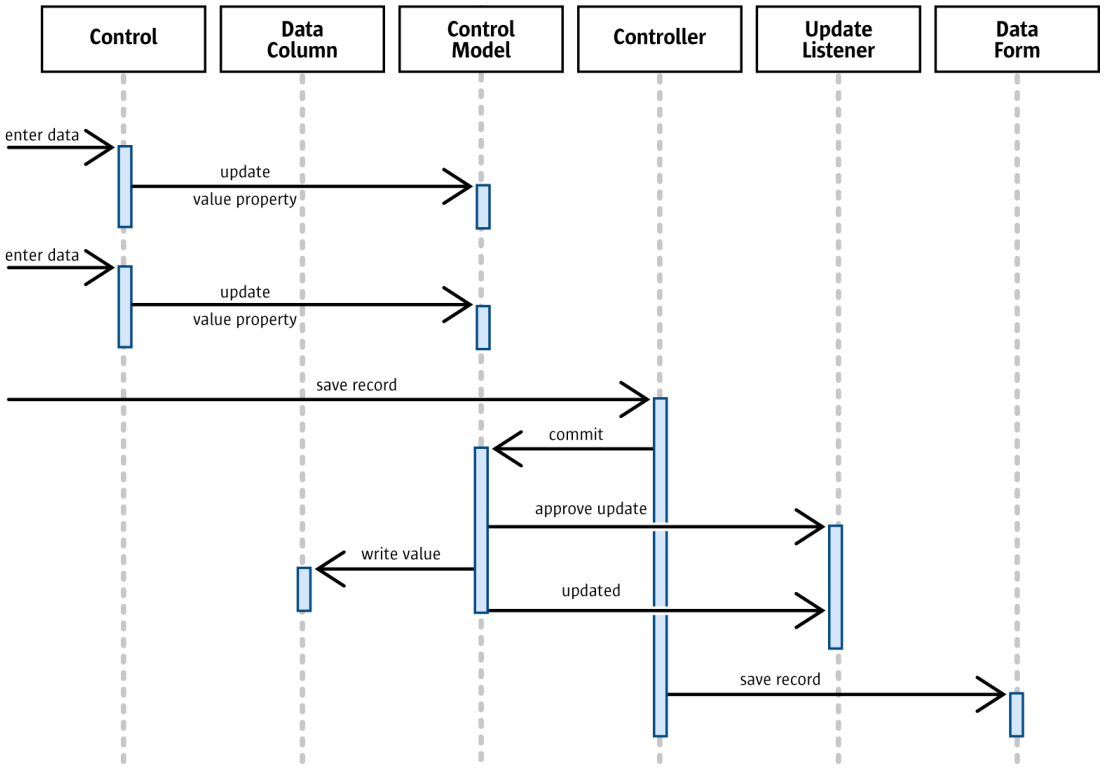


Illustration 198

Note that in the diagram, there is a controller instance involved. In general, this is any instance capable of controlling the user-form interaction. In OpenOffice.org, for every document view and form, there is an instance of the `com.sun.star.form.FormController` service, together with some not-yet UNO-based code that takes on the role of a controller.

13.6 Common Tasks

This chapter is dedicated to problems that may arise when you are working with (or script) form documents, and cannot be solved by OpenOffice.org's built-in methods, but have a solution in the OpenOffice.org UNO API.

13.6.1 Initializing Bound Controls

All form controls specify a default value that is used when initially displaying the control, and when it is reset. For instance, resetting (`com.sun.star.form.XReset`) happens when a form is moved to the insert row, that allows data to be inserted as a new row into the underlying row set.

Now, you do not want a fixed default value for new records, but a dynamically generated one that is dependent on the actual context at the moment the new record is entered.

Or, you want to have real `null` values for date fields. This is currently not possible, because the `com.sun.star.form.component.DateField` service interprets a `null` default as an instruction to use the current system date. Effectively, you cannot have date fields in forms which default to `null` on new records, but you can get this by programming the API. (Forms/FormLayer.java)

```
public void handleReset(EventObject aEvent) throws com.sun.star.uno.RuntimeException {
```

```
    if (((Boolean)xFormProps.getPropertyValue("IsNew")).booleanValue()) {
        // the form is positioned on the insert row

        Object aModifiedFlag = xFormProps.getPropertyValue("IsModified");

        // get the columns of the form
        XColumnsSupplier xSuppCols = (XColumnsSupplier)UnoRuntime.queryInterface(
            XColumnsSupplier.class, xFormProps);
        XNameAccess xCols = xSuppCols.getColumns();

         // and update the date column with a NULL value
         XColumnUpdate xDateColumn = (XColumnUpdate)UnoRuntime.queryInterface(
             XColumnUpdate.class, xCols.getByName("SALEDATE"));
        xDateColumn.updateNull();

        // then restore the flag
        xFormProps.setPropertyValue("IsModified", aModifiedFlag);
    }
}
```

The first decision is where to step in. We chose to add a reset-listener to the form, so that the form is reset as soon as it has been positioned on the new record. The `com.sun.star.form.XReset:resetted()` method is called after the positioning is done.

However, resets also occur for various reasons therefore check if the form is really positioned on the insert row, indicated by the `IsNew` property being `true`.

Now besides retrieving and updating the data column with the desired value, `null`, there is another obstacle. When the form is moved to the insert row, and some values are initialized, the row should not be modified. This is because a modified row is saved in the database, and we only initialized the new row with the defaults, the user did not enter data., We do not want to store the row, therefore we save and restore the `IsModified` flag on the form while doing the update.

13.6.2 Automatic Key Generation

Another problem frequently encountered is the automatic generation of unique keys. There are reasons for doing this on the client side, and missing support, for example, auto-increment fields in your database backend, or you need this value before inserting the row. OpenOffice.org is currently limited in re-fetching the server-side generated value *after* a record has been inserted.

Assume that you have a method called `generateUniqueKey()` to generate a unique key that could be queried from a key generator on a database server, or in a single-user-environment by selecting the maximum of the existing keys and incrementing it by 1. This fragment inserts the generated value into the given column of a given form: (Forms/KeyGenerator.java)

```
public void insertUniqueKey(XPropertySet xForm, String sFieldName) throws com.sun.star.uno.Exception {
    // get the column object
    XColumnsSupplier xSuppCols = (XColumnsSupplier)UnoRuntime.queryInterface(
        XColumnsSupplier.class, xForm);
    XNameAccess xCols = xSuppCols.getColumns();
    XColumnUpdate xCol = (XColumnUpdate)UnoRuntime.queryInterface(
        XColumnUpdate.class, xCols.getByName(sFieldName));

    xCol.updateInt(generateUniqueKey(xForm, sFieldName));
}
```

A solution to determine when the insertion is to happen has been introduced in a previous chapter, that is, we could fill in the value as soon as the form is positioned on the insert row, wait for the user's input in the other fields, and save the record.

Another approach is to step in immediately before the record is inserted. For this, the `com.sun.star.sdb.XRowSetApproveBroadcaster` is used. It notifies listeners when rows are inserted, the listeners can veto this, and final changes can be made to the new record: (Forms/KeyGenerator.java)

```
public boolean approveRowChange(RowChangeEvent aEvent) throws com.sun.star.uno.RuntimeException {
    if (RowChangeAction.INSERT == aEvent.Action) {
        // the affected form
```

```
        XPropertySet xFormProps = (XpropertySet)UnoRuntime.queryInterface(
            XpropertySet.class, aEvent.Source);
        // insert a new unique value
        insertUniqueKey(xFormProps, m_sFieldName);
    }
    return true;
}
```

13.6.3 Data Validation

OpenOffice.org's only offering for client-side data validation is that it automatically rejects `null` values for fields where input is required.

Often you want to validate data as soon as it is written. You have two possibilities here:

- From the chapter *13.5.2 Forms - Data Awareness - Data Aware Controls - Committing Controls*, you can approve updates, and veto the changes a control wants to write into the data column it is bound to.
- Additionally, you can step in later. You know how to use a `com.sun.star.sdb.XRowSetApproveListener` for doing last-minute changes to a record that is about to be inserted.
  Additionally, you can use the listener to approve changes to the row set data. As the `com.sun.star.sdb.RowChangeAction` is sent to the listeners, it distinguishes between different kinds of data modification. You can implement listeners that act differently for insertions and simple updates.

Note the important differences between both solutions. Using an `com.sun.star.form.XUpdateListener` implies that the data operations are vetoed for a given control. Your listener is invoked as soon as the respective control is committed, for instance, when it loses the focus. This implies that changes done to the data column by other means than through this control are not monitored.

The second alternative is using an `com.sun.star.sdb.XRowSetApproveListener` meaning you veto changes immediately before they are sent to the database. Thus, it is irrelevant where they have been made previously. In addition, error messages that are raised when the user actively tries to save the record are considered less disturbing than error messages raised when the user simply leaves a control.

The example below shows the handling for denying empty values for a given control: (Forms/GridFieldValidator.java)

```
public boolean approveUpdate(EventObject aEvent) throws com.sun.star.uno.RuntimeException {
    boolean bApproved = true;

    // the control model which fired the event
    XPropertySet xSourceProps = UNO.queryPropertySet(aEvent.Source);

    String sNewText = (String)xSourceProps.getPropertyValue("Text");
    if (0 == sNewText.length()) {
        // say that the value is invalid
        showInvalidValueMessage();
        bApproved = false;

        // reset the control value
        // for this, we take the current value from the row set field the control
        // is bound to, and forward it to the control model
        XColumn xBoundColumn = UNO.queryColumn(xSourceProps.getPropertyValue("BoundField"));
        if (null != xBoundColumn) {
            xSourceProps.setPropertyValue("Text", xBoundColumn.getString());
        }
    }

    return bApproved;
}
```

14 Universal Content Broker

14.1 Overview

14.1.1 Capabilities

The *Universal Content Broker* (UCB) is a key part of the OpenOffice.org architecture. In general, the UCB provides a standard interface for generalized access to different data sources andfunctions for querying, modifying, and creating data contents. The OpenOffice.org document types are all handled by the UCB. In addition, it is used for help files, directory trees and resource links.

The advantage of delegating resource access to the UCB is, that document, folder and link handling can always be the same from the developer's perspective.It does not matter if you are storing in a file system, on an FTPWebDAV server, or in a document management system.

However, the UCB does not have to be used directly if you want to load and save OpenOffice.org documents.The `com.sun.star.frame.Desktop` serviceprovides the necessary functions, hiding the comparably low-level UCB calls . See *6.1.5 Office Development - OpenOffice.org Application Environment - Handling Documents*. The UCB allows you to administer files in a directory tree or read your own document stream, regardless of where the directory tree or the stream is located.

14.1.2 Architecture

Conceptually, the UCB can be pictured as an object system that consists of a core and a set of Universal Content Providers (UCPs). The UCPs are designed to mask the differences between access protocols, enabling developers to focus on the essentials of integrating resources through the UCB interface, instead of the complexities of an underlying protocol. To this end, each UCP implements an interface that facilitates access to a particular data source through a Uniform Resource Identifier (URI). When a client requests a particular resource, it addresses the UCB that calls a qualified UCP, based on the URI that is associated with the content.

As a rule, all data content is encapsulated in content objects. Each content object implements a standard set of interfaces, that includes functions for querying the content type and a select set of commands that can be run on the respective content, such as `"open"`, `"delete"`, and `"move"`.



Whenever we refer to UCB commands, we put them in double quotes as in `"getPropertyValues"` to make a distinction between UCB commands and methods in general, which are written as `getPropertyValues()`. UCB commands are explained in the section *14.4.3 Universal Content Broker - Using the UCB API - Executing Content Commands* below.

Each content object also has a set of attributes that can be read and set by an application, that include the title, the media type (MIME type), and different flags. The UCB API defines a set of standard commands and properties. There is a set of mandatory properties and commands that must be supported by any content implementation, as well as optional commands and properties with predefined semantics. Illustration 199 shows the relationship between the UCB, UCPs and UCB content objects.

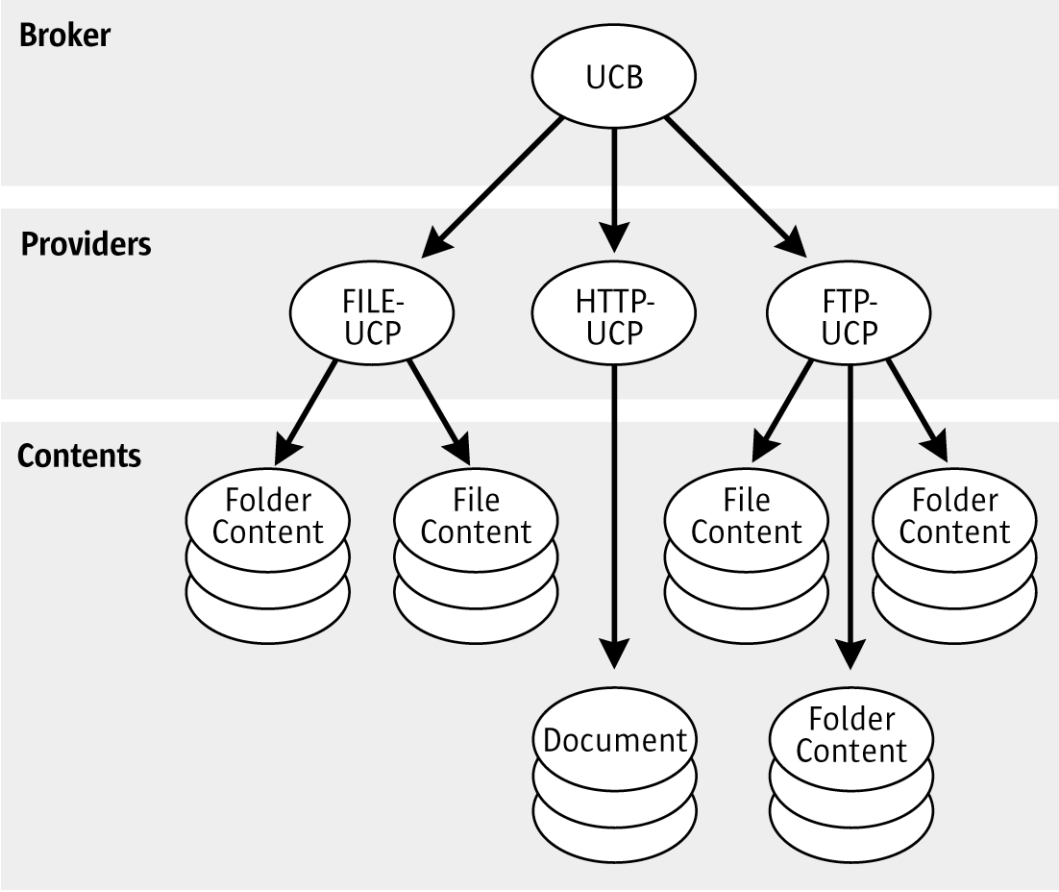


Illustration 199

When a client requests a particular content, it addresses the UCB and passes on the corresponding URI. The UCB analyzes the URI and then calls the corresponding UCP which creates an object for the requested resource.

For example, when an application requests a particular document, the URI of the document is passed to the Universal Content Broker. The UCB analyzes the URI and delegates it to the appropriate UCP. The UCP creates a content object for the requested resource and returns it to the UCB, which returns it to the application. The application now opens the content object or query, or set property values by executing the appropriate command.

14.2 Services and Interfaces

Each UCB content implements the service `com.sun.star.ucb.Content`. The UCB content service interfaces include:

- `com.sun.star.ucb.XContent`
- `com.sun.star.beans.XPropertyContainer`
- `com.sun.star.container.XChild (optional)`

- com.sun.star.ucb.XCommandProcessor
- com.sun.star.ucb.XCommandProcessor2 (optional)
- com.sun.star.ucb.XContentCreator (optional)

The interface com.sun.star.ucb.XContent provides access to a content's type and identifier. The com.sun.star.ucb.XCommandProcessor executes commands at the content object, such as opening a contentthat provides access to the content's data stream or its children, andsetting and getting property values. The interface com.sun.star.beans.XPropertyContainer adds new properties to a content or removes properties that were previously added using this interface. The properties added are always made persistent.



If you change the set of properties by adding or removing properties, the cache of scripting languages, such as OpenOffice.org Basic might not reflect these changes. Thus, use the get/set methods to access the properties in scripting langugages rather than relying on their automatic recognition of properties.

The com.sun.star.ucb.XContentCreator interface is for creating new resources, such as a new folder in the local file system. Not all content implementation can create new resources, therefore this interface is optional. The optional interface com.sun.star.container.XChild provides access to the content object's parent content object. Not all data sources represented by content implementations are organized hierarchically, therefore a parent cannot always be specified.



The interface com.sun.star.ucb.XCommandProcessor2 is the improved version of com.sun.star.ucb.XCommandProcessor. It has been introduced to release command identifiers retrieved through createCommandIdentifier() at the XCommandProcessor interface. To avoid resource leaks, use XCommandProcessor2.

Some content commands defined by the UCB API are listed in the following table:

Selected Command Names for com.sun.star.ucb.XCommandProcessor	
"getCommandInfo"	Obtains an interface that queries information on commands supported by a content.
"getPropertySetInfo"	Obtains an interface that queries information on properties supported by a content.
"getPropertyValues"	Obtains property values from the content.
"setPropertyValues"	Sets property values of the content.
"open"	Gives access to the data stream of a document or to the children of a folder.
"delete"	Destroys a resource.
"insert"	Commits newly-created resources. Writes new data stream of existing document resources.
"transfer"	Copies or moves a content object.

Some interesting content properties defined by the UCB API:

Selected Properties of UCB Contents	
ContentType	Contains a unique(!), read-only type string for the content, for example, "application/vnd.sun.star.hierarchy-link". This is not the Media-Type!
IsFolder	Indicates whether a content can contain other contents.
IsDocument	Indicates whether a content is a document.
Title	Contains the title of an object, for example, the name of a file.
DateCreated	Contains the date and time the object was created.
DateModified	Contains the date and time the object was last modified.
MediaType	Contains the media type (MIME type) of a content.

Selected Properties of UCB Contents	
`Size`	Contains the size, usually in bytes, of an object.

Every UCP implements the service `com.sun.star.ucb.ContentProvider`. The UCP core interface is `com.sun.star.ucb.XContentProvider`. This interface facilitates the creation of content objects based on a given content identifier.

A UCB implements the service `com.sun.star.ucb.UniversalContentBroker`. The UCB core interfaces are `com.sun.star.ucb.XContentProvider` and `com.sun.star.ucb.XContentProviderManager`. The `com.sun.star.ucb.XContentProvider` interface implementation delegates requests to create content objects to the content provider registered for the supplied content identifier. The `com.sun.star.ucb.XContentProviderManager` interface is used to query the UCPs registered with a given UCB, and to register and remove UCPs.



A specification for the implementation for each of the UCPs, including URL schemes, content types, supported commands and properties is located in *C Appendix - Universal Content Providers*.

14.3 Content Providers

The current implementation of the Universal Content Broker in a OpenOffice.org installation supplies UCPs for the following data sources:

Data source	Description	URL Schema	Service name
FILE	Provides access to the file system	"file"	com.sun.star.ucb.FileContentProvider
WebDAV and HTTP	Provides access to web-based file systems and includes HTTP	"vnd.sun.star.webdav" or "http"	com.sun.star.ucb.WebDAVContentProvider
FTP	Provides access to file transfer protocol servers	"ftp"	com.sun.star.ucb.fpx.ContentProvider
Hierarchy	Virtual hierarchy of folders and links	"vnd.sun.star.hier"	com.sun.star.ucb.HierarchyContentProvider
ZIP and JAR files	Packaged files	"vnd.sun.star.pkg"	com.sun.star.ucb.PackageContentProvider
Help files	OpenOffice.org help system	"vnd.sun.star.help"	com.sun.star.help.XMLHelp

Appendix *C Appendix - Universal Content Providers* describes all theabove content providers in more detail. The reference documentation for the commands and other features of these UCPs are located in the SDK or the ucb project on *ucb.openoffice.org*. Additionally, the ucb project offers information about other UCPs for OpenOffice.org, for example, a UCP for document management systems.

14.4 Using the UCB API

This section explains how to use the API of the Universal Content Broker.

14.4.1 Instantiating the UCB

The following steps have to be performed before a process can use the UCB:

- Create and set the UNO service manager.
- Create an instance of the UNO service `com.sun.star.ucb.UniversalContentBroker`, passing the keys identifying a predefined UCB configuration to `createInstanceWithArguments()`.

There are several predefined UCB configurations. Each configuration contains data that describes a set of UCPs. All UCPs contained in a configuration are registered at the UCB that is created using this configuration. A UCB configuration is identified by two keys that are strings. The standard configuration is "`Local`" and "`Office`", which generally allows access to all UCPs available in a local installation.

```
import com.sun.star.lang.XMultiServiceFactory;
import com.sun.star.uno.Exception;
import com.sun.star.uno.XInterface;

boolean initUCB() {

    ////////////////////////////////////////////////////////////////////////
    // Obtain Process Service Manager.
    ////////////////////////////////////////////////////////////////////////

    XMultiServiceFactory xServiceFactory = ...

    ////////////////////////////////////////////////////////////////////////
    // Create UCB. This needs to be done only once per process.
    ////////////////////////////////////////////////////////////////////////

    XInterface xUCB;
    try {
        // Supply configuration to use for this UCB instance...
        String[] keys = new String[2];
        keys[ 0 ] = "Local";
        keys[ 0 ] = "Office";

        xUCB = xServiceFactory.createInstanceWithArguments(
           "com.sun.star.ucb.UniversalContentBroker", keys );
    }
    catch (com.sun.star.uno.Exception e) {
    }

    if (xUCB == null)
        return false;

    return true;
}
```

For information about other configurations, refer to *14.5 Universal Content Broker - UCB Configuration*.

14.4.2 Accessing a UCB Content

Each UCB content can be identified using a URL that points to a folder or a document content in the data source you want to work with. To create a content object for a given URL:

1. Obtain access to the UCB.
2. Let the UCB create a content identifier object for the requested URL using `createContentIdentifier()` at the `com.sun.star.ucb.XContentIdentifierFactory` of the UCB.
3. Let the UCB create a content object for the content identifier using `queryContent()` at the `com.sun.star.ucb.XContentProvider` interface of the UCB.

The UCB selects a UCP according to the URL contained in the identifier object and dispatches the `queryContent()` call to it. The UCP creates the content implementation object and returns it to the UCB, which passes it on to the caller.

Creating a UCB content from a given URL: (UCB/Helper.java)

```
import com.sun.star.uno.UnoRuntime;
import com.sun.star.uno.Xinterface;
import com.sun.star.ucb.*;

{
    String aURL = ...

    ////////////////////////////////////////////////////////////////////////
    // Obtain access to UCB...
    ////////////////////////////////////////////////////////////////////////

    XInterface xUCB = ...

    // Obtain required UCB interfaces XContentIdentifierFactory and XContentProvider
    XContentIdentifierFactory xIdFactory = (XContentIdentifierFactory)UnoRuntime.queryInterface(
        XContentIdentifierFactory.class, xUCB);
    XContentProvider xProvider = (XContentProvider)UnoRuntime.queryInterface(
        XContentProvider.class, xUCB);

    ////////////////////////////////////////////////////////////////////////
    // Obtain content object from UCB...
    ////////////////////////////////////////////////////////////////////////

    // Create identifier object for given URL.
    XContentIdentifier xId = xIdFactory.createContentIdentifier(aURL);

    XContent xContent = xProvider.queryContent(xId);
}
```

14.4.3 Executing Content Commands

Each UCB content is able to execute commands. When the content object is created, commands are executed using its `com.sun.star.ucb.XCommandProcessor` interface. The `execute()` method at this interface expects a `com.sun.star.ucb.Command`, which is a struct containing the command name, command arguments and a handle:

Members of struct `com.sun.star.ucb.Command`	
`Name`	string, contains the name of the command
`Handle`	long, contains an implementation-specific handle for the command
`Argument`	any, contains the argument of the command

Refer to appendix *C Appendix - Universal Content Providers* for a complete list of predefined commands, , the description of the UNO service `com.sun.star.ucb.Content` and the UCP reference that comes with the SDK. For the latest information, visit *ucb.openoffice.org*.



Whenever we refer to UCB commands, we put them in double quotes as in "`getPropertyValues`" to make a distinction between UCB commands and methods in general that are written `getPropertyValues()`.

If executing a command cannot proceed because of an error condition, the following occurs. If the execute call was supplied with a `com.sun.star.ucb.XCommandEnvironment` that contains a `com.sun.star.task.XInteractionHandler`, this interaction handler is used to resolve the problem. If no interaction handler is supplied by passing `null` to the `execute()` method, or it cannot resolve the problem, an exception describing the error condition is thrown.

The following method `executeCommand()` executes a command at a UCB content: (UCB/Helper.java)

```
import com.sun.star.uno.UnoRuntime;
import com.sun.star.ucb.*;

Object executeCommand(XContent xContent, String aCommandName, Object aArgument)
    throws com.sun.star.ucb.CommandAbortedException, com.sun.star.uno.Exception {

    ////////////////////////////////////////////////////////////////////////
```

```
    // Obtain command processor interface from given content.
    ////////////////////////////////////////////////////////////////////////

    XCommandProcessor xCmdProcessor = (XCommandProcessor)UnoRuntime.queryInterface(
        XCommandProcessor.class, xContent);

    ////////////////////////////////////////////////////////////////////////
    // Assemble command to execute.
    ////////////////////////////////////////////////////////////////////////

    Command aCommand  = new Command();
    aCommand.Name     = aCommandName;
    aCommand.Handle   = -1; // not available
    aCommand.Argument = aArgument;

    // Note: throws CommandAbortedException and Exception since
    // we pass null for the XCommandEnvironment parameter
    return xCmdProcessor.execute(aCommand, 0, null);
}
```



The method `executeCommand()` from the example above is used in the following examples whenever a command is to be executed at a UCB content.

14.4.4 Obtaining Content Properties

A UCB content maintains a set of properties. It supports the command `"getPropertyValues"`, that obtains one or more property values from a content. This command takes a sequence of `com.sun.star.beans.Property` and returns an implementation of the interface `com.sun.star.sdbc.XRow`that is similar to a row of a JDBC resultset. To obtain property values from a UCB content:

1. Define a sequence of properties you want to obtain the values for.
2. Let the UCB content execute the command `"getPropertyValues"`.
3. Obtain the property values from the returned row object.

The following example demonstrates the use of content properties. Note that the method `executeCommand()` is used from the example above to execute the `"getPropertyValues"` command that takes a command name and creates a `com.sun.star.ucb.Command` struct from it: (UCB/PropertiesRetriever.java)

```
import com.sun.star.ucb.*;
import com.sun.star.sdbc.XRow;
import com.sun.star.beans.Property;

{
    XContent xContent = ...

    ////////////////////////////////////////////////////////////////////////
    // Obtain value of the string property Title and the boolean property
    // IsFolder from xContent...
    ////////////////////////////////////////////////////////////////////////

    // Define property sequence.

    Property[] aProps = new Property[2];
    Property prop1 = new Property();
    prop1.Name = "Title";
    prop1.Handle = -1;   // n/a
    aProps[0] = prop1;
    Property prop2 = new Property();
    prop2.Name = "IsFolder";
    prop2.Handle = -1;   // n/a
    aProps[1] = prop2;

    XRow xValues;
    try {
        // Execute command "getPropertyValues"
        // using helper method executeCommand (see 14.4.3 Universal Content Broker - Using the UCB API -
Executing Content Commands)
        xValues = executeCommand(xContent, "getPropertyValues", aProps);
    }
```

```
    catch (com.sun.star.ucb.CommandAbortedException e) {
        ... error ...
    }
    catch ( com.sun.star.uno.Exception e ) {
        ... error ...
    }

    // Extract values from row object. Note that the
    // first column is 1, not 0.

    // Title: Obtain value of column 1 as string.
    String aTitle = xValues.getString(1));
    if (aTitle.length() == 0 && xValues.wasNull())
        ... error ...

    // IsFolder: Obtain value of column 2 as boolean.
    boolean bFolder = xValues.getBoolean(2);
    if (!bFolder && xValues.wasNull())
        ... error ...
}
```

The returned row for the content above has two columns Title and IsFolder, and could contain the following data. The column values are retrieved using the getXXX methods of the `com.sun.star.sdbc.XRow` interface. The command `"getPropertyValues"` always returns a single row for contents.

Title	IsFolder
"MyFolder"	TRUE

14.4.5 Setting Content Properties

A UCB content maintains a set of properties. It supports the command `"setPropertyValues"`, that is used to set one or more property values of a content. This command takes a sequence of `com.sun.star.beans.PropertyValue` and returns `void`. To set property values of a UCB content:

- Define a sequence of property values you want to set.
- Let the UCB content execute the command `"setPropertyValues"`.

Note that the command is not aborted if one or more of the property values cannot be set, because the requested property is not supported by the content or because it is read-only. Currently, there is no other methodto check if a property value was set successfully other than to obtain the property value after a set-operation. This may change when status information could be returned by the command `"setPropertyValues"`.

Setting property values of a UCB content: (UCB/PropertiesComposer.java)

```
import com.sun.star.ucb.*;
import com.sun.star.beans.PropertyValue;

{
    XContent xContent = ...
    String aNewTitle = "NewTitle";

    ////////////////////////////////////////////////////////////////////
    // Set value of the string property Title...
    ////////////////////////////////////////////////////////////////////

    // Define property value sequence.

    PropertyValue[] aProps = new PropertyValue[ 1 ];
    PropertyValue aProp = new PropertyValue();
    aProp.Name   = "Title";
    aProp.Handle = -1;   // n/a
    aProp.Value  = aNewTitle;
    aProps[0] = aProp;

    try {
        // Execute command "setPropertyValues".
```

```
        // using helper method executeCommand (see 14.4.3 Universal Content Broker - Using the UCB API -
Executing Content Commands).
        executeCommand(xContent, "setPropertyValues", aProps);
    }
    catch (com.sun.star.ucb.CommandAbortedException e) {
        ... error ...
    }
    catch (com.sun.star.uno.Exception e) {
        ... error ...
    }
}
```

14.4.6 Folders

Accessing the Children of a Folder

A UCB content that is a folder, that is, the value of the required property `IsFolder` is true, supports the command `"open"`. This command takes an argument of type `com.sun.star.ucb.OpenCommandArgument2`. The value returned is an implementation of the service `com.sun.star.ucb.DynamicResultSet`. This `DynamicResultSet` holds the children of the folder and is a result set that can notify registered listeners about changes. To retrieve data from it, call `getStaticResultSet()` at its `com.sun.star.ucb.XDynamicResultSet` interface. The static result set is a `com.sun.star.sdbc.XResultSet` that can be seen as a table, where each row contains a child content of the folder. Use the appropriate methods of `com.sun.star.sdbc.XResultSet` to navigate through the rows:

```
boolean first()
boolean last()
boolean next()
boolean previous()
boolean absolute( [in] long row)
boolean relative( [in] long rows)
void afterLast()
void beforeFirst()
boolean isBeforeFirst()
boolean isAfterLast()
boolean isFirst()
boolean isLast()
long getRow()
```

The child contents are accessed by travelling to the appropriate row and using the interface `com.sun.star.ucb.XContentAccess`, which is implemented by the returned result set:

```
com::sun::star::ucb::XContent queryContent()
string queryContentIdentifierString()
com::sun::star::ucb::XContentIdentifier queryContentIdentifier()
```

You may supply a sequence of `com.sun.star.beans.Property` as part of the argument of the `"open"` command. In this case, the resultset contains one column for each property value that is requested. The property values are accessed by travelling to the appropriate row and calling methods of the interface `com.sun.star.sdbc.XRow`. Refer to the documentation of `com.sun.star.ucb.OpenCommandArgument2` for more information about other parameters that can be passed to the “open” command.

To access the children of a UCB content:

1. Fill the `com.sun.star.ucb.OpenCommandArgument2` structure according to your requirements.
2. Let the UCB content execute the `"open"` command.
3. Access the children and the requested property values using the returned dynamic result set.

Accessing the children of a UCB folder content: (UCB/ChildrenRetriever.java)

```
import com.sun.star.uno.UnoRuntime;
import com.sun.star.ucb.*;
```

```
import com.sun.star.sdbc.XResultSet;
import com.sun.star.sdbc.XRow;

{
    XContent xContent = ...

    ////////////////////////////////////////////////////////////////////
    // Open a folder content, request property values for the string
    // property Title and the boolean property IsFolder...
    ////////////////////////////////////////////////////////////////////
    // Fill argument structure...

    OpenCommandArgument2 aArg = new OpenCommandArgument2();
    aArg.Mode = OpenMode.ALL;     // FOLDER, DOCUMENTS -> simple filter
    aArg.Priority = 32768;        // Ignored by most implementations

    // Fill info for the properties wanted.
    Property[] aProps = new Property[2];
    Property prop1 = new Property();
    prop1.Name = "Title";
    prop1.Handle = -1;   // n/a
    aProps[0] = prop1;
    Property prop2 = new Property();
    prop2.Name = "IsFolder";
    prop2.Handle = -1;   // n/a
    aProps[1] = prop2;

    aArg.Properties = aProps;

    XDynamicResultSet xSet;
    try {
        // Execute command "open".
        // using helper method executeCommand (see 14.4.3 Universal Content Broker - Using the UCB API -
Executing Content Commands.
        xSet = executeCommand(xContent, "open", aArg);
    }
    catch (com.sun.star.ucb.CommandAbortedException e) {
        ... error ...
    }
    catch (com.sun.star.uno.Exception e) {
        ... error ...
    }

    XResultSet xResultSet = xSet.getStaticResultSet();

    ////////////////////////////////////////////////////////////////////
    // Iterate over children, access children and property values...
    ////////////////////////////////////////////////////////////////////

    try {
        // Move to begin.
        if (xResultSet.first()) {
            // obtain XContentAccess interface for child content access and XRow for properties
            XContentAccess xContentAccess = (XContentAccess)UnoRuntime.queryInterface(
                XContentAccess.class, xResultSet);
            XRow xRow = (XRow)UnoRuntime.queryInterface(XRow.class, xResultSet);
            do {
                // Obtain URL of child.
                String aId = xContentAccess.queryContentIdentifierString();

                // First column: Title (column numbers are 1-based!)
                String aTitle = xRow.getString(1);
                if (aTitle.length() == 0 && xRow.wasNull())
                    ... error ...

                // Second column: IsFolder
                boolean bFolder = xRow.getBoolean(2);
                if (!bFolder && xRow.wasNull())
                    ... error ...
            } while (xResultSet.next()) // next child
        }
    }
    catch (com.sun.star.ucb.ResultSetException e) {
        ... error ...
    }
}
```

14.4.7 Documents

Reading a Document Content

A UCB content that is a document, that is, the value of the required property `IsDocument` is `true`, supportsthe command `"open"`. The command takes an argument of type `com.sun.star.ucb.OpenCommandArgument2`. Note that this command with the same argument type is also used to access the children of a folder. As seen in the examples, the argument's `Mode` member controls access to the children or the data stream, or both for contents that support both. If you are interested in the data stream, ignore the command's return value, which will presumably be a null value.

The caller must implement a data sink and supply this implementation as `"open"` command arguments to get access to the data stream of a document. These data sinks are called back by the implementation when the `"open"` command is executed. There are two different interfaces for data sinks to choose from, `com.sun.star.io.XActiveDataSink` and `com.sun.star.io.XOutputStream`.

- `XActiveDataSink`**:** If this type of data sink is supplied, the caller of the command is active. It consists of the following methods:

```
void setInputStream( [in] com::sun::star::io::XInputStream aStream)
com::sun::star::io::XInputStream getInputStream()
```

The implementation of the command supplies an implementation of the interface `com.sun.star.io.XInputStream` to the given data sink using `setInputStream()` and return. Once the execute-call has returned, the caller accesses the input stream calling getInputStream() and read the data using that stream, through `readBytes()` or `readSomeBytes()`.

- `XOutputStream`**:** If this type of data sink is supplied, the caller of the command is passive. The data sink is called back through the following methods of `XOutputStream`:

```
void writeBytes( [in] sequence< byte > aData)
void closeOutput()
void flush()
```

The implementation of the command writes all data to the output stream calling `writeBytes()` and closes it through `closeOutput()` after all data was successfully written. Only then will the open command return.

The type to use depends on the logic of the client application. If the application is designed so that it passively processes the data supplied by an `com.sun.star.io.XOutputStream` using an output stream as sink is advantageous, because many content providers implement this case efficiently without buffering any data. If the application is designed so that it actively reads the data, use an `com.sun.star.io.XActiveDataSink`, then any necessary buffering takes place in the implementation of the open command.

The following example shows a possible implementation of an `com.sun.star.io.XActiveDataSink` and its usage with the `"open"` command: (UCB/MyActiveDataSink.java)

```
import com.sun.star.io.XActiveDataSink;
import com.sun.star.io.XInputStream;

//////////////////////////////////////////////////////////////////////////
// XActiveDataSink interface implementation.
//////////////////////////////////////////////////////////////////////////

public class MyActiveDataSink implements XActiveDataSink {
    XInputStream m_aStream = null;

    public MyActiveDataSink() {
        super();
```

```
    }

    public void setInputStream(XInputStream aStream) {
        m_aStream = aStream;
    }

    public XInputStream getInputStream() {
        return m_aStream;
    }
};
```

Now this data sink implementation can be used with the "`open`" command. After opening the document content, `getInputStream()` returns the input stream containing the document data. The actual byte content is read using `readSomeBytes()` in the following fragment: (UCB/DataStreamRetriever.java)

```
import com.sun.star.ucb.*;
import ...MyActiveDataSink;

{
    XContent xContent = ...

    ////////////////////////////////////////////////////////////////////////
    // Read the document data stream of a document content using a
    // XActiveDataSink implementation as data sink....
    ////////////////////////////////////////////////////////////////////////
    // Fill argument structure...

    OpenCommandArgument2 aArg = new OpenCommandArgument2;
    aArg.Mode = OpenMode.DOCUMENT;
    aArg.Priority = 32768;        // Ignored by most implementations

    // Create data sink implementation object.
    XActiveDataSink xDataSink = new MyActiveDataSink;
    aArg.Sink = xDataSink;

    try {
        // Execute command "open". The implementation of the command will
        // supply an XInputStream implementation to the data sink,
        // using helper method executeCommand (see 14.4.3 Universal Content Broker - Using the UCB API -
Executing Content Commands)
        executeCommand(xContent, "open", aArg);
    }
    catch (com.sun.star.ucb.CommandAbortedException e) {
        ... error ...
    }
    catch (com.sun.star.uno.Exception e) {
        ... error ...
    }

    // Get input stream supplied by the open command implementation.
    XInputStream xData = xDataSink.getInputStream();
    if (xData == null)
        ... error ...

    ////////////////////////////////////////////////////////////////////////
    // Read data from input stream...
    ////////////////////////////////////////////////////////////////////////
    try {
        // Data buffer. Will be allocated by input stream implementation!
        byte[][] aBuffer = new byte[1][1];

        int nRead = xData.readSomeBytes(aBuffer, 65536);
        while (nRead > 0) {
            // Process data contained in buffer.
            ...

            nRead = xData.readSomeBytes(aBuffer, 65536);
        }

        // EOF.
    }
    catch (com.sun.star.io.NotConnectedException e) {
        ... error ...
    }
    catch (com.sun.star.io.BufferSizeExceededException e) {
        ... error ...
    }
    catch (com.sun.star.io.IOException e) {
        ... error ...
    }
}
```

Storing a Document Content

A UCB content that is a document, that is, the value of the required property `IsDocument` is `true`, supports the command `"insert"`. This command is used to overwrite the document's data stream. The command requires an argument of type `com.sun.star.ucb.InsertCommandArgument` and returns `void`. The caller supplies the implementation of an `com.sun.star.io.XInputStream` with the command argument. This stream contains the data to be written. An additional flag indicating if an existing content and its data should be overwritten is supplied with the command argument. Implementations that are not able to detect if there are previous data may ignore this parameter and will always write the new data.

Setting or storing the content data stream of a UCB document content is shown below: (UCB/DataStreamComposer.java)

```
import com.sun.star.ucb.*;
import com.sun.star.io.XInputStream;

{
    XContent xContent = ...
    XInputStream xData = ... // The data to write.

    ////////////////////////////////////////////////////////////////////
    // Write the document data stream of a document content...
    ////////////////////////////////////////////////////////////////////

    // Fill argument structure...

    InsertCommandArgument aArg = new InsertCommandArgument();
    aArg.Data = xData;
    aArg.ReplaceExisting = true;

    try {
        // Execute command "insert".
        // using helper method executeCommand (see 14.4.3 Universal Content Broker - Using the UCB API -
Executing Content Commands).
        executeCommand(xContent, "insert", aArg);
    }
    catch (com.sun.star.ucb.CommandAbortedException e) {
        ... error ...
    }
    catch (com.sun.star.uno.Exception e) {
        ... error ...
    }
}
```

14.4.8 Managing Contents

Creating

A UCB content that implements the interface `com.sun.star.ucb.XContentCreator` acts as a factory for new resources. For example, a file system folder can be a creator for other file system folders and files.

A new content object created by the `com.sun.star.ucb.XContentCreator` implementation can be considered as an *empty hull* for a content object of a special type. This new content object has to be filled with some property values to become fully functional. For example, a file system folder could require a name, represented by the property `Title` in the UCB. The interface `com.sun.star.ucb.XContentCreator` offers ways to determine what contents can be created and what properties need to be set. Information can be obtained on the general type, such as `FOLDER`, `DOCUMENT`, or `LINK`, of the objects. After the required property values are set, the creation process needs to be committed by using the command `"insert"`. Note that this command is always executed by the new content, not by the content creator, because the creator is not necessarily the parent of the new content. The flag `ReplaceExisting` in the `"insert"` argument

`com.sun.star.ucb.InsertCommandArgument` usually is `false`, because the caller does not want to destroy an already existing resource. The `"insert"` command implementation makes the new content persistent in the appropriate storage medium.

To create a new resource:

1. Obtain the interface `com.sun.star.ucb.XContentCreator` from a suitable UCB content.
2. Call `createNewContent()` at the content creator. Supply information on the type of content to create with the arguments. The argument expected is a `com.sun.star.ucb.ContentInfo` struct.
3. Obtain and set the property values that are mandatory for the content just created.
4. Let the new content execute the command `"insert"` to complete the creation process.

Creating a new resource: (UCB/ResourceCreator.java)

```
import com.sun.star.uno.UnoRuntime;
import com.sun.star.ucb.*;
import com.sun.star.beans.PropertyValue;
import com.sun.star.io.XInputStream;

{
    XContent xContent = ...

    ////////////////////////////////////////////////////////////////////////
    // Create a new file system file object...
    ////////////////////////////////////////////////////////////////////////

    // Obtain content creator interface.
    XContentCreator xCreator = (XContentCreator)UnoRuntime.queryInterface(
        XContentCreator.class, xContent);

    // Note: The data for aInfo may have been obtained using
    //       XContentCreator::queryCreatableContentsInfo().
    //       A number of possible types is listed below

    ContentInfo aInfo = new ContentInfo();
    aInfo.Type = "application/vnd.sun.staroffice.fsys-file";
    aInfo.Attributes = 0;

    // Create new, empty content.
    XContent xNewContent = xCreator.createNewContent(aInfo);

    if (xNewContent == null)
        ... error ...

    ////////////////////////////////////////////////////////////////////////
    // Set mandatory properties...
    ////////////////////////////////////////////////////////////////////////

    // Obtain a name for the new file.
    String aFilename = ...

    // Define property value sequence.
    PropertyValue[] aProps = new PropertyValue[1];
    PropertyValue aProp = new PropertyValue;
    aProp.Name   = "Title";
    aProp.Handle = -1; // n/a
    aProp.Value  = aFilename;
    aProps[ 0 ] = aProp;
    try {
        // Execute command "setPropertyValues".
        // using helper method executeCommand (see 14.4.3 Universal Content Broker - Using the UCB API -
Executing Content Commands)
        executeCommand(xNewContent, "setPropertyValues",aProps);
    }
    catch (com.sun.star.ucb.CommandAbortedException e) {
        ... error ...
    }
    catch (com.sun.star.uno.Exception e) {
        ... error ...
    }

    ////////////////////////////////////////////////////////////////////////
    // Write the new file to disk...
    ////////////////////////////////////////////////////////////////////////

    // Obtain document data for the new file.
```

```
    XInputStream xData = ...

    // Fill argument structure...
    InsertCommandArgument aArg = new InsertCommandArgument();
    aArg.Data = xData;
    aArg.ReplaceExisting = false;

    try {
        // Execute command "insert".
        executeCommand(xNewContent, "insert", aArg);
    }
    catch (com.sun.star.ucb.CommandAbortedException e) {
        ... error ...
    }
    catch (com.sun.star.uno.Exception e) {
        ... error ...
    }
}
```

The appendix *C Appendix - Universal Content Providers* discusses the creation of contents for all available UCPs. The table below shows a number of `com.sun.star.ucb.ContentInfo` types for creatable contents. Additionally, you can ask the content creator for its creatable contents using `com.sun.star.ucb.XContentCreator:queryCreatableContentsInfo()`. The UCB reference in the SDK and on *ucb.openoffice.org* offers comprehensive information about creatable contents.

Data source	Content Info Type	Content	Content Service that Creates the Contents
FILE	"application/vnd.sun.staroffice.fsys-folder"	folder	com.sun.star.ucb.FileContent
	"application/vnd.sun.staroffice.fsys-file"	document	
WebDAV and HTTP	"application/vnd.sun.star.webdav-collection"	folder	com.sun.star.ucb.WebDAVFolderContent
	"application/http-content"	document	
FTP	"application/vnd.sun.staroffice.ftp-folder"	folder	com.sun.star.ucb.ChaosContent
	"application/vnd.sun.staroffice.ftp-file"	document	
Hier-archy	"application/vnd.sun.star.hier-folder"	folder	com.sun.star.ucb.HierarchyFolderContent
	"application/vnd.sun.star.hier-link"	Link	
ZIP and JAR files	"application/vnd.sun.star.pkg-folder"	folder	com.sun.star.ucb.PackageFolderContent
	"application/vnd.sun.star.pkg-stream"	document	

Deleting

Executing the command `"delete"` on a UCB content destroys the resource it represents. This command takes a boolean parameter. If it is set to `true`, the resource is immediately, destroyed physically.



The command also destroys all existing sub-resources of the resource to be destroyed!

If `false` is passed to this command, the caller wants to delete the resource "logically". This means that the resource is restored or physically destroyed later. A soft-deleted content needs to support the command `"undelete"`. This command brings it back to life. The implementation of the delete command can ignore the parameter and may opt to always destroy the resource physically.



Currently we do not have a trash service that could be used by UCB clients to manage soft-deleted contents.

Deleting a resource: (UCB/ResourceRemover.java)

```
import com.sun.star.ucb.*;

{
    XContent xContent = ...

    ////////////////////////////////////////////////////////////////////
    // Destroy a resource physically...
    ////////////////////////////////////////////////////////////////////

    try {
        Boolean bDeletePhysically = new Boolean(true);

        // Execute command "delete".
        // using helper method executeCommand (see 14.4.3 Universal Content Broker - Using the UCB API -
Executing Content Commands)
        executeCommand(xContent, "delete", bDeletePhysically);
    }
    catch (com.sun.star.ucb.CommandAbortedException e) {
        ... error ...
    }
    catch (com.sun.star.uno.Exception e) {
        ... error ...
    }
}
```

Copying, Moving and Linking

Copying, moving and creating links to a resource works differently from the other operations available for UCB Contents. There are *three UCB Contents* involved in these operations, the source object, target folder, and target object. There may even be *two content Providers,* for example, when moving a file located on an FTP server to the local file system of a workstation. Each implementation of the `com.sun.star.ucb.UniversalContentBroker` service supports the `com.sun.star.ucb.XCommandProcessor` interface. This command processor implements the command `"globalTransfer"` that can be used to copy and move UCB Contents, and create links to UCB Contents. The command takes an argument of type `com.sun.star.ucb.GlobalTransferCommandArgument`. To copy, move or create a link to a resource, execute the `"globalTransfer"` command at the UCB.



The reasons for the different handling are mainly technical. We did not want to force every single implementation of the transfer command of a UCB content to accept nearly all types of contents. Instead, we wanted to have one single implementation that would be able to handle all types of contents.

Copying, moving and creating links to a resource are shown in the following example: (UCB/ResourceManager.java)

```
import com.sun.star.ucb.*;
import com.sun.star.uno.UnoRuntime;
import com.sun.star.uno.XInterface;

{
    String aSourceURL = ...       // URL of the source object
    String aTargetFolderURL = ... // URL of the target folder

    ////////////////////////////////////////////////////////////////////
    // Obtain access to UCB...
    ////////////////////////////////////////////////////////////////////
    XInterface xUCB = ...

    // Obtain XCommandProcessor interface from UCB...
    XCommandProcessor xProcessor = (XCommandProcessor)UnoRuntime.queryInterface(
        XCommandProcessor.class, xUCB);

    if (xProcessor == null)
        ... error ...
    ////////////////////////////////////////////////////////////////////
    // Copy a resource to another location...
    ////////////////////////////////////////////////////////////////////
    try {
        GlobalTransferCommandArgument aArg = new GlobalTransferCommandArgument();
        aArg.TransferCommandOperation = TransferCommandOperation_COPY;
        aArg.SourceURL = aSourceURL;
```

```
        aArg.TargetURL = aTargetFolderURL;

        // object keeps it current name
        aArg.NewTitle = "";

        // fail, if object with same name exists in target folder
        aArg.NameClash = NameClash.ERROR;

        // Let UCB execute the command "globalTransfer",
        // using helper method executeCommand (see 14.4.3 Universal Content Broker - Using the UCB API -
Executing Content Commands)
        executeCommand(xProcessor, "globalTransfer", aArg);
    }
    catch (com.sun.star.ucb.CommandAbortedException e) {
        ... error ...
    }
    catch (com.sun.star.uno.Exception e) {
        ... error ...
    }
}
```

UCB Configuration

This section describes how to configure the Universal Content Broker (UCB). Before a process uses the UCB, it needs to configure the UCB. Configuring the UCB means registering a set of Universal Content Providers (UCPs) at a content broker instance. Only UCPs known to the UCB are used to provide content. Generally we provide two ways to configure a UCB:

- Create a default UCB with no UCPs registered and register all required UCPs manually.
- Define a UCB configuration and create a UCB that is automatically configured with the UCPs given in that configuration.

14.4.9 UCP Registration Information

Before registering a content provider, the following information that is provided by the implementer of the UCP is required. The Appendix *C Appendix - Universal Content Providers* provides these for the currently available UCPs.

- The *UNO service name* to instantiate the UCP, for example, "com.sun.star.ucb.FileContentProvider". Each UCP must be implemented and registered as a UNO component. Refer to chapter *4 Writing UNO Components* for more information on writing and registering UNO components.
- An *URL template* used by the UCB to select the registered UCPs that best fit an incoming URL. See `com.sun.star.ucb.XContentIdentifier`. This can be the name of an URL scheme, for example, the file that selects the given UCP for all file URLs, or a limited regular expression, such as `"http://"[^/?#]*".com"([/?#].*)?` That will select the given UCP for all http URLs in the com domain. See the documentation of `com.sun.star.ucb.XContentProviderManager:registerContentProvider()` for details about these regular expressions.
- *Additional arguments* that may be needed by the UCP.

14.4.10 Unconfigured UCBs

A UCB is called *unconfigured* if it has no content providers, thus it is not able to provide any contents. Each UCB implements the interface `com.sun.star.ucb.XContentProviderManager`. This interface offers the functionality to register UCPs at runtime.

To create an unconfigured UCB and configure it manually:

1. Create an instance of the UNO service `com.sun.star.ucb.UniversalContentBroker`.
2. Register the appropriate UCPs using the `com.sun.star.ucb.XContentProviderManager` interface of the UCB.

XContentProviderManager contains the following methods:

```
com::sun::star::ucb::XContentProvider registerContentProvider(
                [in] com::sun::star::ucb::XContentProvider Provider,
                [in] string Scheme,
                [in] boolean ReplaceExisting)
oneway void deregisterContentProvider(
                [in] com::sun::star::ucb::XContentProvider Provider,
                [in] string Scheme)
sequence< com::sun::star::ucb::ContentProviderInfo > queryContentProviders()
com::sun::star::ucb::XContentProvider queryContentProvider([in] string URL)
```

The XContentProvider configures a UCB for content providers, obtains `com.sun.star.ucb.ContentProviderInfo` structs describing the available providers, and the provider that is currently registered for a specific URL schema. The following example uses `registerContentProvider()` to configure an unconfigured UCB for a file content provider.

Unconfigured UCB:

```
import com.sun.star.lang.XMultiServiceFactory;
import com.sun.star.ucb.DuplicateProviderException;
import com.sun.star.ucb.XContentProvider;
import com.sun.star.ucb.XContentProviderManager;
import com.sun.star.uno.Exception;
import com.sun.star.uno.UnoRuntime;

boolean initUCB() {

    ////////////////////////////////////////////////////////////////////////
    // Obtain Process Service Manager.
    ////////////////////////////////////////////////////////////////////////

    XMultiServiceFactory xServiceFactory = ...

    ////////////////////////////////////////////////////////////////////////
    // Create UCB. This needs to be done only once per process.
    ////////////////////////////////////////////////////////////////////////

    XContentProviderManager xUCB;
    try {
        xUCB = (XContentProviderManager)UnoRuntime.queryInterface(
            XContentProviderManager.class, xServiceFactory.createInstance(
                "com.sun.star.ucb.UniversalContentBroker"));
    }
    catch (com.sun.star.uno.Exception e) {
    }

    if (xUCB == null)
        return false;

    ////////////////////////////////////////////////////////////////////////
    // Instanciate UCPs and register at UCB.
    ////////////////////////////////////////////////////////////////////////

    XContentProvider xFileProvider;
    try {
        xFileProvider = (XContentProvider)UnoRuntime.queryInterface(
            XContentProvider.class, xServiceFactory.createInstance(
                "com.sun.star.ucb.FileContentProvider"));
    }
    catch (com.sun.star.uno.Exception e) {
    }

    if (xFileProvider == null)
        return false;

    try {
        // Parameters: provider, URL scheme, boolean flag replaceExisting
        xUCB.registerContentProvider(xFileProvider, "file", new Boolean(false));
    }
    catch (DuplicateProviderException ex) {
    }

    // Create/register other UCPs...

    return true;
```

```
}
```

14.4.11 Preconfigured UCBs

A UCB is called *preconfigured* if it was given a UCB configuration at the time it was instantiated. A UCB configuration contains a set of UCP registration information.

To create a preconfigured UCB:

1. Create an instance of the UNO service `com.sun.star.ucb.UniversalContentBroker`.
2. Pass the configuration as a parameters to the creation function. The UCB instance returned offers all UCPs defined in the given configuration.

Preconfigured UCB:

```
import com.sun.star.lang.XMultiServiceFactory;
import com.sun.star.uno.Exception;
import com.sun.star.uno.XInterface;

boolean initUCB() {
    ////////////////////////////////////////////////////////////////////////
    // Obtain Process Service Manager.
    ////////////////////////////////////////////////////////////////////////

    XMultiServiceFactory xServiceFactory = ...

    ////////////////////////////////////////////////////////////////////////
    // Create UCB. This needs to be done only once per process.
    ////////////////////////////////////////////////////////////////////////

    XInterface xUCB;
    try {
        // Supply configuration to use for this UCB instance...
        String[] keys = new String[2];
        keys[0] = "Local";
        keys[0] = "Office";

        xUCB = xServiceFactory.createInstanceWithArguments(
           "com.sun.star.ucb.UniversalContentBroker", keys);
    }
    catch (com.sun.star.uno.Exception e) {
    }

    if (xUCB == null)
        return false;

    return true;
}
```

A UCB configuration used by a preconfigured UCB describes a set of UCPs available in a configuration. All UCPs contained in a configuration are registered at the UCB that is created using this configuration. A UCB configuration is identified by two keys that are strings. The keys allow some structuring in the configuration files), but they do not have a purpose. See the example file below. The standard configuration is "Local" and "Office", that allows access to all UCPs. The XML sample below shows how these keys are used to organize UCB configurations.

The predefined configurations for OpenOffice.org are defined in the file *<OfficePath>/share/config/data/org/openoffice/ucb/Configuration.xcd*. This file must be adapted to add configurations or edit existing configurations. The XCD file is used during the OpenOffice.org build process to generate the appropriate XML file. This XML file is part of a OpenOffice.org installation and is located in *<OfficePath>share/config/registry/instance/org/openoffice/ucb/Configuration.xml*. The UCB tries to get configuration data from this XML file.

UCB Configuration (*org/openoffice/ucb/Configuration.xcd*):

```
<!DOCTYPE schema:package SYSTEM "../schema/schema.description.dtd">
<schema:package package-id="org.openoffice.ucb.Configuration" xml:lang="en-US"
xmlns:schema="http://openoffice.org/2000/registry/schema/description"
```

```
xmlns:default="http://openoffice.org/2000/registry/schema/default"
xmlns:cfg="http://openoffice.org/2000/registry/instance">

<schema:templates template-id="org.openoffice.ucb.Configuration">

<!-- ContentProvider -->
<schema:group cfg:name="ContentProviderData">
<schema:value cfg:name="ServiceName" cfg:type="string">
</schema:value>
<schema:value cfg:name="URLTemplate" cfg:type="string">
</schema:value>
<schema:value cfg:name="Arguments" cfg:type="string">
</schema:value>
</schema:group>

<!-- ContentProvidersDataSecondaryKeys -->
<schema:group cfg:name="ContentProvidersDataSecondaryKeys">
<schema:set cfg:name="ProviderData"
        cfg:element-type="ContentProviderData"/>
</schema:group>

<!-- ContentProvidersDataPrimaryKeys -->
<schema:group cfg:name="ContentProvidersDataPrimaryKeys">
<schema:set cfg:name="SecondaryKeys"
        cfg:element-type="ContentProvidersDataSecondaryKeys"/>
</schema:group>
</schema:templates>

<schema:component cfg:writable="true"
component-id="org.openoffice.ucb.Configuration"
cfg:notified="true" cfg:localized="false">
<schema:set cfg:name="ContentProviders"
        cfg:element-type="ContentProvidersDataPrimaryKeys">
<default:group cfg:name="Local">
        <default:set cfg:name="SecondaryKeys"
        cfg:element-type="ContentProvidersDataSecondaryKeys">
        <default:group cfg:name="Office">
        <default:set cfg:name="ProviderData"
                cfg:element-type="ContentProviderData">

        <!-- Hierarchy UCP -->
        <default:group cfg:name="Provider1">
                <default:value cfg:name="ServiceName" cfg:type="string">
                <default:data>com.sun.star.ucb.HierarchyContentProvider</default:data>
                </default:value>
                <default:value cfg:name="URLTemplate" cfg:type="string">
                <default:data>vnd.sun.star.hier</default:data>
                </default:value>
                <default:value cfg:name="Arguments" cfg:type="string">
                <default:data/>
                </default:value>
        </default:group>

        <!-- File UCP -->
        <default:group cfg:name="Provider2">
                <default:value cfg:name="ServiceName" cfg:type="string">
                <default:data>com.sun.star.ucb.FileContentProvider</default:data>
                </default:value>
                <default:value cfg:name="URLTemplate" cfg:type="string">
                <default:data>file</default:data>
                </default:value>
                <default:value cfg:name="Arguments" cfg:type="string">
                <default:data/>
                </default:value>
        </default:group>

        <!-- Other UCPs go here -->

        </default:set>
        </default:group>
        </default:set>
</default:group>
</schema:set>
</schema:component>
</schema:package>
```

14.4.12 Content Provider Proxies

The UNO service implementing a UCP must be instantiated at the time the content provider is registered at the UCB. This is done using `com.sun.star.ucb.XContentProviderManager`'s

`registerContentProvider()` method. In some cases, this can consume resources, because instantiating a UNO service means loading the libraries containing its code. As a convention, each UNO component should reside in its own library.

Therefore, a special UNO service is offered that provides a generic proxy for a UCP. Its purpose is to delay the loading of the real UCP code until it is needed. Generally, this does not happen before the first `createContentIdentifier()`/ `queryContent()` calls are done at the proxy.

Instead of registering the real instantiated UCP at the UCB, a proxy is created for the UCP. The UCP registration information is passed to the proxy. The proxy only uses this information to instantiate the real UCP on demand. There is almost no performance overhead with this mechanism.



When using preconfigured UCBs, the UCB implementation uses proxies instead of the real UCPs to avoid wasting resources.

15 Configuration Management

15.1 Overview

15.1.1 Capabilities

The OpenOffice.org configuration management component provides a uniform interface to get and set OpenOffice.org configuration data in an organized manner, independent of the physical data store used for the data.

Currently, the configuration API can only be used to get and set existing configuration options. You can not extend the configuration by new settings for your own purposes. For details, see *15.5 Configuration Management - Customizing Configuration Data.*

15.1.2 Architecture

OpenOffice.org configuration data describes the state or environment of a UNO component or the OpenOffice.org application. There are different kinds of configuration data:

- Static configuration: This is data that describes the configuration of the software and hardware environment. This data is set by a setup tool and does not change at runtime. An example of static configuration data is information about installed filters.
- Explicit settings: This is preference data that can be controlled by the user explicitly. There is a dedicated UI to change these settings. An example explicit settings are the settings controlled through the Tools – Options dialogs in OpenOffice.org.
- Implicit settings: This is status information that is also controlled by the user, but the user does not change explicitly. The application tracks this state in the background, making it persistent across application sessions. An example implicit settings are window positions and states, or a list of the recently used documents.

This list is not comprehensive, but indicates the range of data characteristically stored by configuration management.

The configuration management component organizes the configuration data in a hierarchical structure. The hierarchical structure and the names and data types of entries in this database are described by a schema. Only data that conforms to one of the installed schemas is stored in the database.

The hierarchical database stores any hierarchical information that can be described as a configuration schema, but it is optimized to work with the data needed for application configuration and preferences. Small data items having a well-defined data type are supported efficiently, whereas large, unspecific binary objects should not be stored in the configuration database. These objects are stored in separate files and the configuration keeps the URLs of these files only.

Configuration schemas are provided by the authors of applications and components that use the data. When a component is installed, the corresponding configuration schemas are installed into the configuration management system.

Configuration data is stored in a backend data store. In OpenOffice.org, the standard backend consists of XML files stored in a directory hierarchy. Support for more backends is planned for a future release.

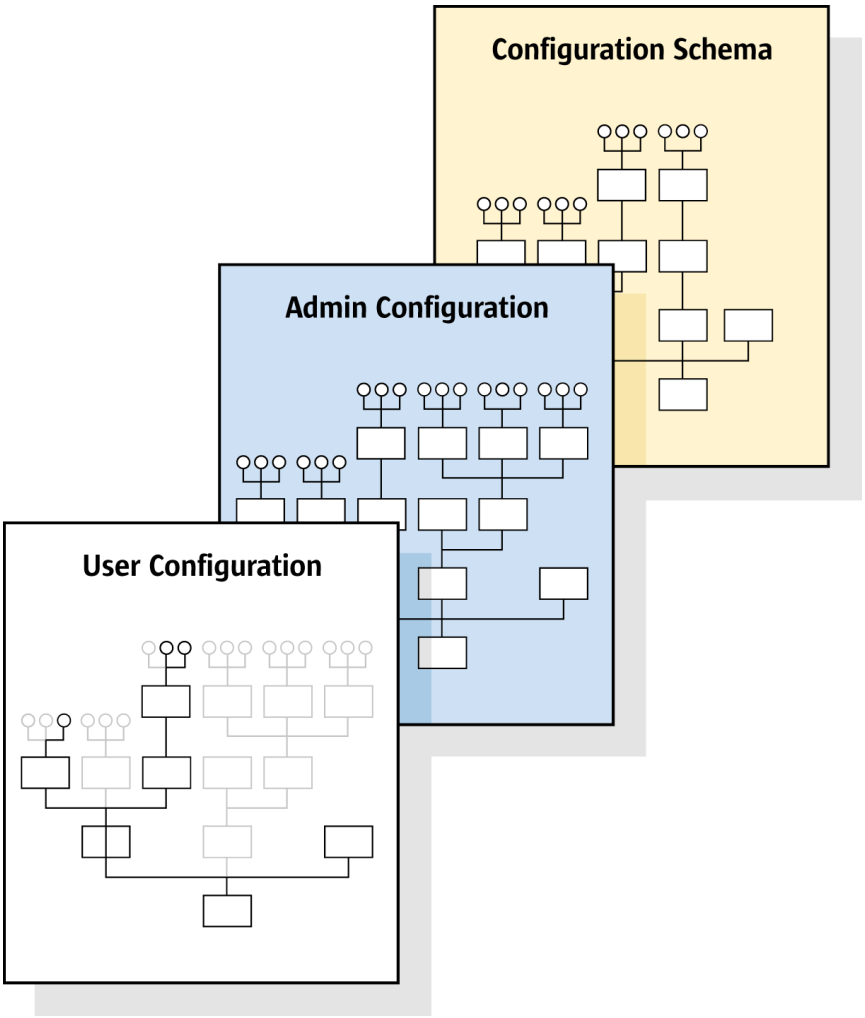


Illustration 200: Configuration layers

For a given schema, multiple layers of data may exist that are merged together at runtime. One or more of these layers define default settings, possibly shared by several users. Additionally, there is a layer specific to a single user and contains personal preferences overriding the shared settings. In normal operations all changes to data affect only this user-specific layer.

Access to the merged configuration data for a user is managed by a `com.sun.star.configuration.ConfigurationProvider` object to connect to a data source, fetch and store data, and merge layers.

This provider provides views on the configuration data. A view is a subset of the entire configuration that can be navigated as an object hierarchy. The objects in this hierarchy represent nodes of the configuration hierarchy to navigate to other nodes and access values of data items.

All configuration items have a fixed type and a name.

The type is prescribed by the schema. The following kinds of items are available:

- 'Properties' are *data* items that contain a single data value or an array of values from a limited set of basic types.
- 'Groups' are *structural* nodesthat contain a collection of child items of various types. The number and names of children, as well as their types, are *fixed* by the schema. Structural and data items can be mixed within one group.
- 'Sets' are *structural* nodesthat serve as *dynamic* containers for a variable number of elements. These elements must be all data or all structural items, and they must all be uniform. In the first case, all values have the same basic type, and in the latter case, all the sub-trees have the same structure. The schema contains templates for container elements, which are prototypes of the element structure.

Properties hold the actual data. Group nodes form the structural skeleton defined in the schema. Set nodes are used to dynamically add and remove configuration data within the confines of the schema. Taken together, they can be used to build hierarchical structures of arbitrary complexity.

Each configuration item has a namethat uniquely identifies the item within its parent, that is, the node in the hierarchical tree that immediately contains the item under consideration. Paths spanning multiple levels of the hierarchy are built similarly to UNIX file system paths. The separator for individual name components in paths is a forward slash ('/'). Paths that begin with a slash are considered 'absolute paths' and must completely specify the location of an item within the hierarchy. Paths that start directly with a name are relative paths and describe the location of an item within one of its ancestors in the hierarchy.

The top-level subdivisions of the configuration hierarchy are called configuration modules. Each configuration module has a schema that describes the data items available within that module. Modules are the unit of schema installation. The name of a configuration module must be globally unique. The names of configuration modules have an internal hierarchical structure using a dot ('.') as a separator, similar to Java package names. The predefined configuration modules of OpenOffice.org use package names from the super-package "org.openoffice.*".

The names of container elements are specified when data items are added to a container. Data item names in the schema are limited to ASCII letters, digits and a few punctuation marks, but there are no restrictions applied to the names of container elements. This requires special handling when referring to a container element in a path. A path component addressing a container element takes the form `<template-pattern>['<escaped-name>']`. Here `<template-pattern>` can be the name of the template describing the element or an asterisk "*" to match any template. The `<escaped-name>` is a representation of the name of the element where a few forbidden characters are represented in an escaped form borrowed from XML. The quotes delimiting the `<escaped-name>` may alternatively be double quote characters "". The following character escapes are used:

Character	Escape
&	&
"	"
'	'

In the table below, are some escaped forms for invented entries in the Set node `/org.openoffice.Office.TypeDetection/Filters` for (fictitious) filters:

Filter Name	Path Component
Plain Text	Filter['Plain Text']
Q & A Book	*["Q & A Book"]
Bob's Filter	*['Bob's Filter']

The UIName value of the last example filter would have an absolute path of /org.openoffice.Office.TypeDetection/Filters/Filter['Bob's Filter']/UIName.

In several places in the configuration management, API arguments are passed to a newly created object instance as Sequence, for example, in the argument to com.sun.star.lang.XMultiServiceFactory:createInstanceWithArguments. Such arguments are type com.sun.star.beans.PropertyValue, and only the fields Name and Value need to be filled.



In the future ,a transition to the more appropriate argument type com.sun.star.beans.NamedValue is planned.

15.2 Object Model

The centralized entry point for configuration access is a com.sun.star.configuration.ConfigurationProvider object. This object represents a connection to a single configuration data source providing access to configuration data for a single user.

The com.sun.star.configuration.AdministrationProvider service is an extended version of this service that enables administrative access to shared configuration data.

The com.sun.star.configuration.ConfigurationProvider serves as a factory for configuration views. A configuration view provides access to the structure and data of a subset of the configuration database. This subset is accessible as a hierarchical object tree. When creating a configuration view, parameters are provided that describe the subset of the data to retrieve. In the simplest case, the only argument is an absolute configuration path that identifies a structural configuration item. This parameter is given as an argument named "nodepath". The configuration view then encompasses the sub-tree which is rooted in the indicated item.

A configuration view is not represented by a single object, but as an object hierarchy formed by all the items that are part of the selected sub-tree. The object that comes closest to representing the view as a whole is the root element of that tree. This object is the one returned by the factory method of the com.sun.star.configuration.ConfigurationProvider. In addition to the simple node-oriented interfaces, it also supports interfaces that apply to the view as a whole.

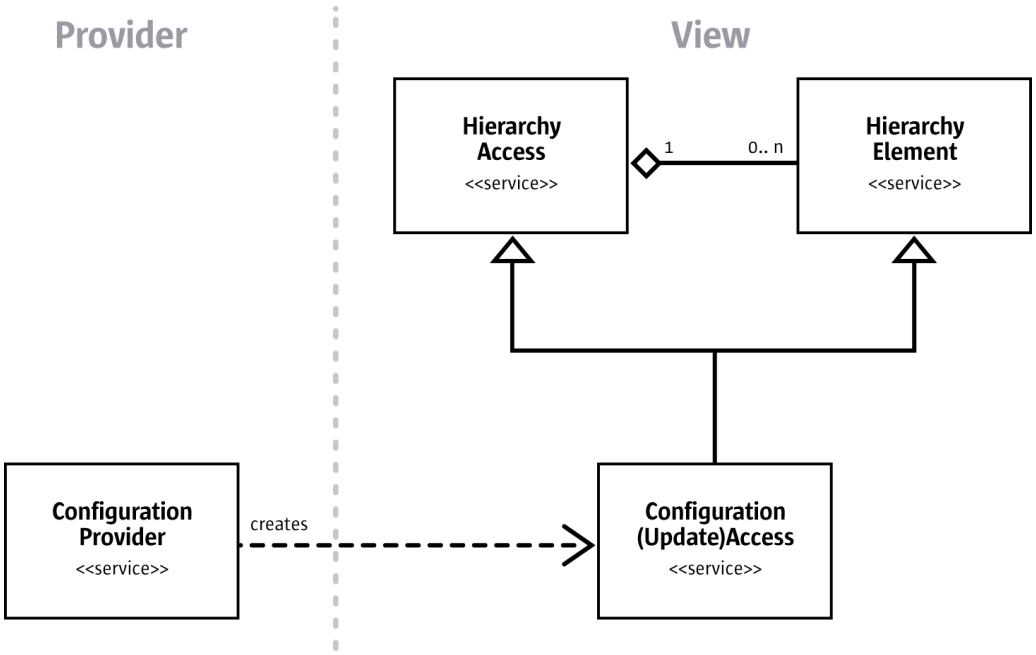


Illustration 201: Configuration object model overview

Within a configuration view, UNO objects with access interfaces are used to represent all structural items. Value items are not represented as objects, but retrieved as types, usually wrapped inside an `any`.

The following types are supported for data items:

`string`	Plain Text (Sequence of [printable] Unicode characters)
`boolean`	Boolean value (true/false)
`short`	16-bit integer number
`int`	32-bit integer number
`long`	64-bit integer number
`double`	Floating point number
`binary`	Sequence of uninterpreted octets

Value items contain a single value, or a sequence or array of one of the basic types. The arrays must be homogeneous, that is, mixed arrays are not supported. The configuration API treats array types as atomic items, there is no built-in support for accessing or modifying individual array elements.



Binary values should be used only for small chunks of data that cannot easily be stored elsewhere. For large BLOBs it is recommended to store links, for example, as URLs, in the configuration.

For example, bitmaps for small icons might be stored in the configuration, whereas large images are stored externally.

All of the structural objects implement the service `com.sun.star.configuration.ConfigurationAccess` that specifies interfaces to navigate the hierarchy and access values within the view. Different instances of this service support different sets of interfaces. The interfaces that an object supports depends on its structural type, that is, is it a group or a set, and context, that is, is it a group member, an element of a set or the root of the view.

A configuration view can be read-only or updatable. This is determined by the access requested when creating the view, but updatability may also be restricted by access rights specified in the schema or data. The basic `com.sun.star.configuration.ConfigurationAccess` service specifies read-only operations. If an object is part of an updatable view and is not marked read-only in the schema or the data, it implements the extended service `com.sun.star.configuration.ConfigurationUpdateAccess`. This service adds interfaces to change values and modify set nodes.

These service names are also used to create the configuration views. To create a view for read access, call `com.sun.star.lang.XMultiServiceFactory:createInstanceWithArguments` at the `com.sun.star.configuration.ConfigurationProvider` to request a `com.sun.star.configuration.ConfigurationAccess`. To obtain an updatable view, the service `com.sun.star.configuration.ConfigurationUpdateAccess` must be requested.

The `com.sun.star.configuration.AdministrationProvider` supports the same service specifiers, but creates views on shared layers of configuration data.

The object initially returned when creating a configuration view represents the root node of the view. The choice of services and interfaces it supports depends on the type of configuration item it represents. The root object has additional interfaces pertaining to the view as a whole. For example, updates of configuration data within a view are combined into batches of related changes, which exhibit transaction-like behavior. This functionality is controlled by the root object of the view.

15.3 Configuration Data Sources

Creating a view to configuration data is a two-step process.

1. Connect to a data source by creating an instance of a `com.sun.star.configuration.ConfigurationProvider` for user preferences or a `com.sun.star.configuration.AdministrationProvider` for shared preferences.
2. Ask the provider to create an access object for a specific nodepath in the configuration database using `com.sun.star.lang.XMultiServiceFactory:createInstanceWithArguments()`. The access object can be a `com.sun.star.configuration.ConfigurationAccess` or a `com.sun.star.configuration.ConfigurationUpdateAccess`.

15.3.1 Connecting to a Data Source

The first step to access the configuration database is to connect to a configuration data source.

To obtain a provider instance ask the global `com.sun.star.lang.ServiceManager` for a `com.sun.star.configuration.ConfigurationProvider`. Typically the first lines of code to get access to configuration data look similar to the following: (Config/ConfigExamples.java)

```
// get my global service manager
XMultiServiceFactory xServiceManager = (XMultiServiceFactory)UnoRuntime.queryInterface(
    XMultiServiceFactory.class, this.getRemoteServiceManager(
        "uno:socket,host=localhost,port=8100;urp;StarOffice.ServiceManager"));

final String sProviderService = "com.sun.star.configuration.ConfigurationProvider";

// create the provider and remember it as a XMultiServiceFactory
XMultiServiceFactory xProvider = (XMultiServiceFactory)
    UnoRuntime.queryInterface(XMultiServiceFactory.class,
                              xServiceManager.createInstance(sProviderService));
```

This code creates a default com.sun.star.configuration.ConfigurationProvider. The most important interface a com.sun.star.configuration.ConfigurationProvider implements is com.sun.star.lang.XMultiServiceFactory that is used to create further configuration objects.

The com.sun.star.configuration.ConfigurationProvider always operates in the user mode, accessing data on behalf of the current user and directing updates to the user's personal layer.

For administrative access to manipulate the default layers the com.sun.star.configuration.AdministrationProvider is used. When creating this service, additional parameters can be used that select the layer for updates or that contain credentials used to authorize administrative access. The backend that is used determines which default layers exist, how they are addressed and how administrative access is authorized. The standard file-based backend has a single shared layer only. The files for this layer are located in the *share* directory of the OpenOffice.org installation. To gain administrative access to this layer, no additional parameters are needed. An com.sun.star.configuration.AdministrationProvider for this backend automatically tries to read and write this shared layer. Authorization is done by the operating system based upon file access privileges. The current user requires write privileges in the shared configuration directory if an AdministrationProvider is suppose to update configuration data.

A com.sun.star.configuration.AdministrationProvider is created in the same way as a com.sun.star.configuration.ConfigurationProvider.

```
// get my global service manager
XMultiServiceFactory xServiceManager = getServiceManager();

// get the arguments to use
com.sun.star.beans.PropertyValue aReinitialize = new com.sun.star.beans.PropertyValue()
aReinitialize.Name = "reinitialize"
aReinitialize.Value = new Boolean(true);

Object[] aProviderArguments = new Object[1];
aProviderArguments[0] = aReinitialize;

final String sAdminService = "com.sun.star.configuration.AdministrationProvider";

// create the provider and remember it as a XMultiServiceFactory
XMultiServiceFactory xAdminProvider = (XMultiServiceFactory)
    UnoRuntime.queryInterface(XMultiServiceFactory.class,
                  xServiceManager.createInstanceWithArguments(sAdminService,aProviderArguments));
```

As you see in the example above, the default com.sun.star.configuration.AdministrationProvider supports a special parameter for reinitialization:

Parameter Name	Type	Default	Comments
reinitialize	boolean	false	Discard any cached information from previous runs and regenerate from scratch.

The current implementation maintains a set of cache files containing pre-parsed representations of the configuration data. If the reinitialize parameter is true, these cache files will be recreated from the XML data when the AdministrationProvider is created.

When establishing the connection, specify the parameters that select the backend to use and additional backend-specific parameters to select the data source. When there are no parameters given, the standard configuration backend and data source of the OpenOffice.org installation is used.

The standard values for these parameters may be found in the configuration file *configmgr(.ini|rc)* (.ini on Windows, rc on Unix) in the *program* directory of the OpenOffice.org installation.



The list of available backends and the parameters they support may change in a future release. Using these parameters are normally not necessary and therefore are not recommended.

The following parameter is supported to select the type of backend to use:

Parameter Name	Type	Default	Comments
servertype	string	"local"	Other values are currently not supported in OpenOffice.org.

For the "local" backend, the following parameters are used to select the location of data:

Parameter Name	Type	Default	Comments
sourcepath	string	$(installurl)/share/config/registry	The INI entry is named CFG_BaseDataURL
updatepath	string	$(userurl)/user/config/registry	The INI entry is named CFG_UserDataURL

Arguments can be provided that determine the default behavior of views created through this com.sun.star.configuration.ConfigurationProvider. The following parameters may be used for this purpose:

Parameter Name	Type	Default	Comments
locale	string	The user's locale.	
lazywrite	boolean	true	The INI entry is named enable_async



After the connection is established, creating another com.sun.star.configuration.ConfigurationProvider using the same parameters may return the same object. The default configuration provider obtained when no arguments are given will always be the same object. Be careful not to call com.sun.star.lang.XComponent:dispose() on a shared com.sun.star.configuration.ConfigurationProvider.

15.3.2 Using a Data Source

After a configuration provider is obtained, call com.sun.star.lang.XMultiServiceFactory:createInstanceWithArguments() to create a view on the configuration data.

The following arguments can be specified when creating a view:

Parameter Name	Type	Default	Comments
nodepath	string	-	This parameter is *required*. It contains an absolute path to the root node of the view.
locale	string	The user's locale (or "*")	Using this parameter, specify the locale to be used for selecting locale-dependent values. Use the ISO code for a locale, for example, en-US for U.S. English.
lazywrite	boolean	true	
depth	integer	(unlimited)	This parameter causes the view to be truncated to a specified nesting depth.
nocache	boolean	false	This parameter is deprecated.



If the special value "*" is uused for the `locale` parameter, values for all locales are retrieved. In this case, a locale-dependent property appears as a set item. The items of the set are the values for the different locales. They will have the ISO identifiers of the locales as names.

This mode is the default if you are using an `com.sun.star.configuration.AdministrationProvider`.

It can be used if you want to assign values for different locales in a targeted manner. Usually this is logical in an administration or installation context only.

To create a read-only view on the data, the service `com.sun.star.configuration.ConfigurationAccess` is requested:

```
// Create a specified read-only configuration view
public Object createConfigurationView(String sPath) throws com.sun.star.uno.Exception {
    // get the provider to use
    XMultiServiceFactory xProvider = getProvider();

    // The service name: Need only read access:
    final String sReadOnlyView = "com.sun.star.configuration.ConfigurationAccess";

    // creation arguments: nodepath
    com.sun.star.beans.PropertyValue aPathArgument = new com.sun.star.beans.PropertyValue();
    aPathArgument.Name = "nodepath";
    aPathArgument.Value = sPath;

    Object[] aArguments = new Object[1];
    aArguments[0] = aPathArgument;

    // create the view
    Object xViewRoot = xProvider.createInstanceWithArguments(sReadOnlyView, aArguments);

    return xViewRoot;
}
```

To obtain updatable access, the service `com.sun.star.configuration.ConfigurationUpdateAccess` is requested. In this case, there are additional parameters available that control the caching behavior of the configuration management component:

```
// Create a specified updatable configuration view
Object createUpdatableView(String sPath, boolean bAsync) throws com.sun.star.uno.Exception {
    // get the provider to use
    XMultiServiceFactory xProvider = getProvider();

    // The service name: Need update access:
    final String cUpdatableView = "com.sun.star.configuration.ConfigurationUpdateAccess";

    // creation arguments: nodepath
    com.sun.star.beans.PropertyValue aPathArgument = new com.sun.star.beans.PropertyValue();
    aPathArgument.Name = "nodepath";
    aPathArgument.Value = sPath;

    // creation arguments: commit mode - write-through or write-back
    com.sun.star.beans.PropertyValue aModeArgument = new com.sun.star.beans.PropertyValue();
    aModeArgument.Name = "lazywrite";
    aModeArgument.Value = new Boolean(bAsync);

    Object[] aArguments = new Object[2];
    aArguments[0] = aPathArgument;
    aArguments[1] = aModeArgument;

    // create the view
    Object xViewRoot = xProvider.createInstanceWithArguments(cUpdatableView, aArguments);

    return xViewRoot;
}
```

A `com.sun.star.configuration.AdministrationProvider` supports the same service specifiers, but creates views on shared layers of configuration data. It supports additional parameters to select the exact layer to work on or to specify authorization credentials. For the standard file-based `com.sun.star.configuration.AdministrationProvider`, the parameters are not defined. For a `com.sun.star.configuration.AdministrationProvider`, the default value for the `locale` parameter is "*".

15.4 Accessing Configuration Data

15.4.1 Reading Configuration Data

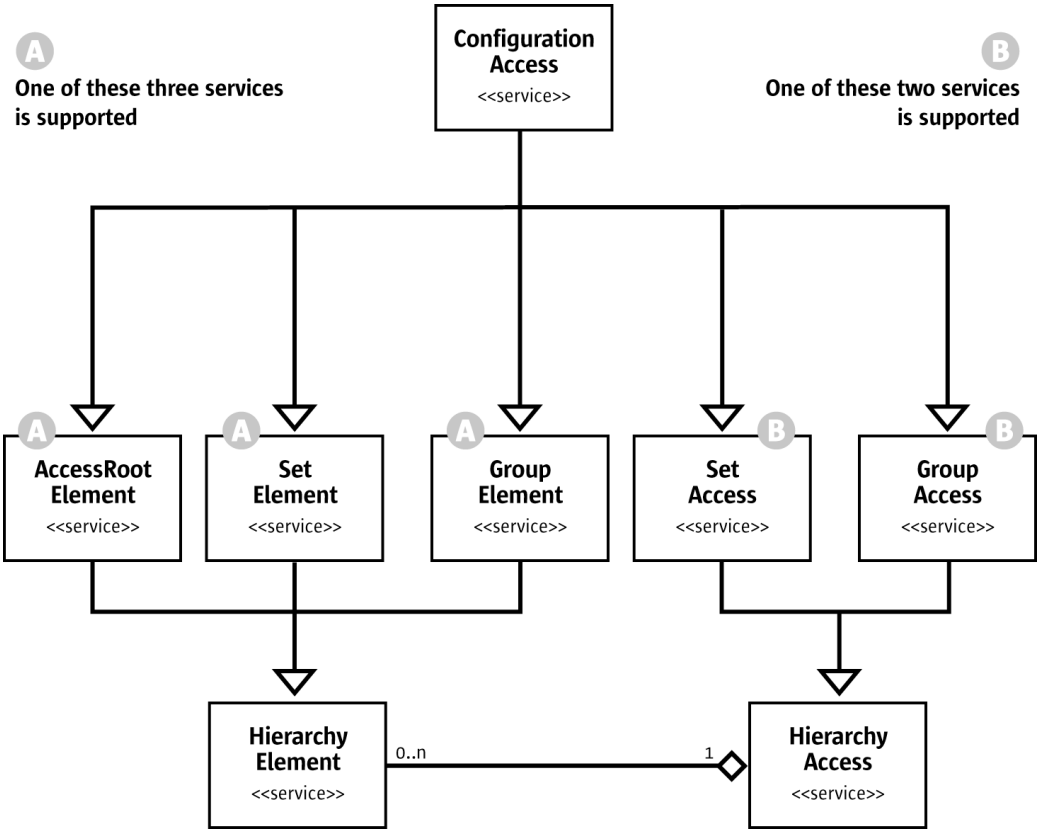


Illustration 202: ConfigurationAccess services

The `com.sun.star.configuration.ConfigurationAccess` service is used to navigate through the configuration hierarchy and reading values. It also provides information about a node and its context.

The following example shows how to collect or display information about a part of the hierarchy. For processing elements and values, our example uses its own callback Java interface `IConfigurationProcessor`:

```
// Interface to processs information when browsing the configuration tree
public interface IConfigurationProcessor {
    // process a value item
   public abstract void processValueElement(String sPath_, Object aValue_);
    // process a structural item
   public abstract void processStructuralElement(String sPath_, XInterface xElement_);
};
```

Then, we define a recursive browser function:

```
// Internal method to browse a structural element recursively in preorder
public void browseElementRecursively(XInterface xElement, IConfigurationProcessor aProcessor)
        throws com.sun.star.uno.Exception {
    // First process this as an element (preorder traversal)
    XHierarchicalName xElementPath = (XHierarchicalName) UnoRuntime.queryInterface(
        XHierarchicalName.class, xElement);

    String sPath = xElementPath.getHierarchicalName();

    //call configuration processor object
```

```
    aProcessor.processStructuralElement(sPath, xElement);

    // now process this as a container of named elements
    XNameAccess xChildAccess =
        (XNameAccess) UnoRuntime.queryInterface(XNameAccess.class, xElement);

    // get a list of child elements
    String[] aElementNames = xChildAccess.getElementNames();

    // and process them one by one
    for (int i=0; i< aElementNames.length; ++i) {
        Object aChild = xChildAccess.getByName(aElementNames[i]);

        // is it a structural element (object) ...
        if ( aChild instanceof XInterface ) {
            // then get an interface
            XInterface xChildElement = (XInterface)aChild;

            // and continue processing child elements recursively
            browseElementRecursively(xChildElement, aProcessor);
        }
        // ... or is it a simple value
        else {
            // Build the path to it from the path of
            // the element and the name of the child
            String sChildPath;
            sChildPath = xElementPath.composeHierarchicalName(aElementNames[i]);

            // and process the value
            aProcessor.processValueElement(sChildPath, aChild);
        }
    }
}
```

Now a driver procedure is defined which uses our previously defined routine `createConfigurationView()` to create a view, and then starts processing:

```
/** Method to browse the part rooted at sRootPath
    of the configuration that the Provider provides.

    All nodes will be processed by the IConfigurationProcessor passed.
*/
public void browseConfiguration(String sRootPath, IConfigurationProcessor aProcessor)
        throws com.sun.star.uno.Exception {

    // create the root element
    XInterface xViewRoot = (XInterface)createConfigurationView(sRootPath);

    // now do the processing
    browseElementRecursively(xViewRoot, aProcessor);

    // we are done with the view - dispose it
    //   This assumes that the processor
    //   does not keep a reference to the elements in processStructuralElement

    ((XComponent) UnoRuntime.queryInterface(XComponent.class,xViewRoot)).dispose();
    xViewRoot = null;
}
```

Finally, as an example of how to put the code to use, the following is code to print the currently registered file filters:

```
/** Method to browse the filter configuration.

    Information about installed filters will be printed.
*/
public void printRegisteredFilters() throws com.sun.star.uno.Exception {
    final String sProviderService = "com.sun.star.configuration.ConfigurationProvider";
    final String sFilterKey = "/org.openoffice.Office.TypeDetection/Filters";

   // browse the configuration, dumping filter information
    browseConfiguration( sFilterKey,
       new IConfigurationProcessor () {  // anonymous implementation of our custom interface
           // prints Path and Value of properties
           public void processValueElement(String sPath_, Object aValue_) {
               System.out.println("\tValue: " + sPath_ + " = " + aValue_);
           }
           // prints the Filter entries
           public void processStructuralElement( String sPath_, XInterface xElement_) {
               // get template information, to detect instances of the 'Filter' template
               XTemplateInstance xInstance =
                   ( XTemplateInstance )UnoRuntime.queryInterface( XTemplateInstance .class,xElement_);

               // only select the Filter entries
```

```
                if (xInstance != null && xInstance.getTemplateName().endsWith("Filter")) {
                     XNamed xNamed = (XNamed)UnoRuntime.queryInterface(XNamed.class,xElement_);
                     System.out.println("Filter " + xNamed.getName() + " (" + sPath_ + ")");
                }
            }
        } );
}
```

For access to sub-nodes, a `com.sun.star.configuration.ConfigurationAccess` supports container interfaces `com.sun.star.container.XNameAccess` and `com.sun.star.container.XChild`. These interfaces access the immediate child nodes in the hierarchy , as well as `com.sun.star.container.XHierarchicalNameAccess` for direct access to items that are nested deeply.

These interfaces are uniformly supported by all structural configuration items. Therefore, they are utilized by code that browses a sub-tree of the configuration in a generic manner.

Parts of the hierarchy where the structure is known statically can also be viewed as representing a complex object composed of properties, that are composed of sub-properties themselves. This model is supported by the interface `com.sun.star.beans.XPropertySet` for child access and `com.sun.star.beans.XHierarchicalPropertySet` for access to deeply nested properties within such parts of the hierarchy. Due to the static nature of property sets, this model does not carry over to set nodes that are dynamic in nature and do not support the associated interfaces.

For effective access to multiple properties, the corresponding `com.sun.star.beans.XMultiPropertySet` and `com.sun.star.beans.XMultiHierarchicalPropertySet` interfaces are supported.



In a read-only view, all properties are marked as `com.sun.star.beans.PropertyAttribute:READONLY` in `com.sun.star.beans.XPropertySetInfo`. Attempts to use `com.sun.star.beans.XPropertySet:setPropertyValue()` to change the value of a property fail accordingly.

Typically, these interfaces are used to access a known set of preferences. The following example reads grid option settings from the OpenOffice.org Calc configuration into this structure:

```
class GridOptions
{
    public boolean visible;
    public int resolution_x;
    public int resolution_y;
    public int subdivision_x;
    public int subdivision_y;
};
```

These data may be read by a procedure such as the following. It demonstrates different approaches to read data:

```
// This method reads information about grid settings
protected GridOptions readGridConfiguration() throws com.sun.star.uno.Exception {
    // The path to the root element
    final String cGridOptionsPath = "/org.openoffice.Office.Calc/Grid";

    // create the view
    Object xViewRoot = createConfigurationView(cGridOptionsPath);

    // the result structure
    GridOptions options = new GridOptions();

    // accessing a single nested value
    // the item /org.openoffice.Office.Calc/Grid/Option/VisibleGrid is a boolean data item
    XHierarchicalPropertySet xProperties =
        (XHierarchicalPropertySet)UnoRuntime.queryInterface(XHierarchicalPropertySet.class, xViewRoot);

    Object aVisible = xProperties.getHierarchicalPropertyValue("Option/VisibleGrid");
    options.visible = ((Boolean) aVisible).booleanValue();

    // accessing a nested object and its subproperties
    // the item /org.openoffice.Office.Calc/Grid/Subdivision has sub-properties XAxis and YAxis
    Object xSubdivision = xProperties.getHierarchicalPropertyValue("Subdivision");
```

```
    XMultiPropertySet xSubdivProperties = (XMultiPropertySet)UnoRuntime.queryInterface(
        XMultiPropertySet.class, xSubdivision);

    // String array containing property names of sub-properties
    String[] aElementNames = new String[2];

    aElementNames[0] = "XAxis";
    aElementNames[1] = "YAxis";

    // getPropertyVAlues() returns an array of any objects according to the input array aElementNames
    Object[] aElementValues = xSubdivProperties.getPropertyValues(aElementNames);

    options.subdivision_x = ((Integer) aElementValues[0]).intValue();
    options.subdivision_y = ((Integer) aElementValues[1]).intValue();

    // accessing deeply nested subproperties
    // the item /org.openoffice.Office.Calc/Grid/Resolution has sub-properties
    // XAxis/Metric and YAxis/Metric
    Object xResolution = xProperties.getHierarchicalPropertyValue("Resolution");

    XMultiHierarchicalPropertySet xResolutionProperties = (XMultiHierarchicalPropertySet)
        UnoRuntime.queryInterface(XMultiHierarchicalPropertySet.class, xResolution);

    aElementNames[0] = "XAxis/Metric";
    aElementNames[1] = "YAxis/Metric";

    aElementValues = xResolutionProperties.getHierarchicalPropertyValues(aElementNames);

    options.resolution_x = ((Integer) aElementValues[0]).intValue();
    options.resolution_y = ((Integer) aElementValues[1]).intValue();

    // all options have been retrieved - clean up and return
    // we are done with the view - dispose it

    ((XComponent)UnoRuntime.queryInterface(XComponent.class, xViewRoot)).dispose();

    return options;
}
```

A `com.sun.star.configuration.ConfigurationAccess` also supports the interfaces `com.sun.star.container.XNamed`, `com.sun.star.container.XHierarchicalName` and `com.sun.star.beans.XPropertySetInfo` to retrieve information about the node, as well as interface `com.sun.star.container.XChild` to get the parent within the hierarchy. To monitor changes to specific items, register listeners at the interfaces `com.sun.star.container.XContainer` and `com.sun.star.beans.XPropertySet`.

The exact set of interfaces supported depends on the role of the node in the hierarchy. For example, a set node does not support `com.sun.star.beans.XPropertySet` and related interfaces, but it supports `com.sun.star.configuration.XTemplateContainer` to get information about the template that specifies the schema of elements. The root object of a configuration view does not support `com.sun.star.container.XChild`, but it supports `com.sun.star.util.XChangesNotifier` to monitor all changes in the whole view.

15.4.2 Updating Configuration Data

A `com.sun.star.configuration.ConfigurationUpdateAccess` provides operations for updating configuration data, by extendingthe interfaces supported by a `com.sun.star.configuration.ConfigurationAccess`.

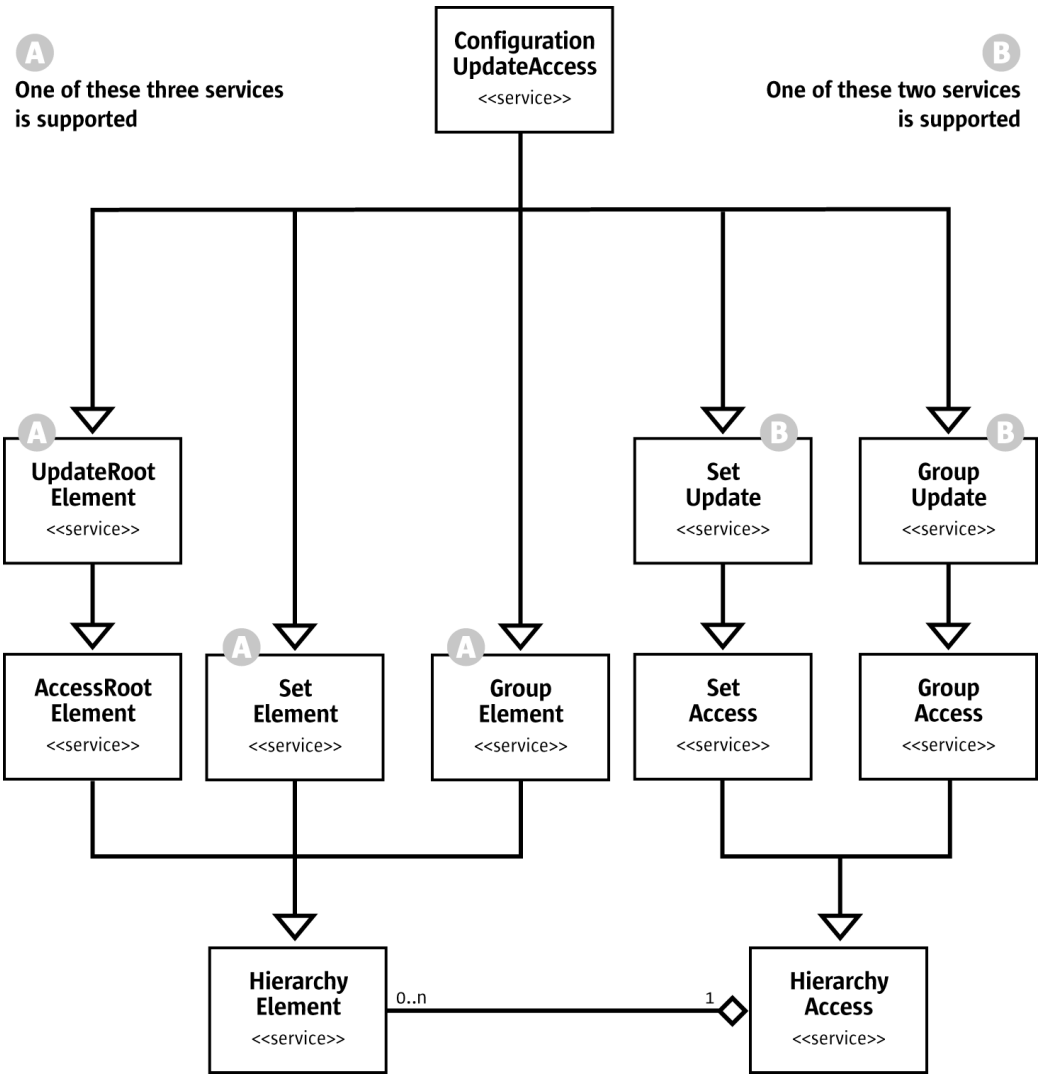


Illustration 203: ConfigurationUpdateAccess services

For `com.sun.star.beans.XPropertySet` and related interfaces, the semantics are extended to set property values. Support for container interfaces is extended to set properties in group nodes, and insert or remove elements in set nodes. Thus, a `com.sun.star.configuration.GroupUpdate` supports interface `com.sun.star.container.XNameReplace` and a `com.sun.star.configuration.SetUpdate` supports `com.sun.star.container.XNameContainer`. Only complete trees having the appropriate structure are inserted for sets whose elements are complete structures as described by a template,. To support this, the set object is used as a factory that can create structures of the appropriate type. For this purpose, the set supports `com.sun.star.lang.XSingleServiceFactory`.

Updates done through a configuration view are only visible within that view, providing transactional isolation. When a set of updates is ready, it must be committed explicitly to become visible beyond this view. All pending updates are then sent to the configuration provider in one batch. This batch update behavior is controlled through interface`com.sun.star.util.XChangesBatch` that is implemented by the root element of an updatable configuration view.



When a set of changes is committed to the provider it becomes visible to other views obtained from the same provider as an atomic and consistent set of changes. Thus, in the local scope of a single `com.sun.star.configuration.ConfigurationProvider` a high degree of transactional behavior is achieved.

The configuration management component does not guarantee true transactional behavior. Committing the changes to the `com.sun.star.configuration.ConfigurationProvider` does not ensure persistence or durability of the changes. When the provider writes back the changes to the persistent data store, they become durable. Generally, the `com.sun.star.configuration.ConfigurationProvider` may cache and combine requests, so that updates are propagated to the data store at a later time.

If several sets of changes are combined before being saved, isolation and consistency may be weakened in case of failure. As long as the backend does not fully support transactions, only parts of an update request might be stored successfully, thus violating atomicity and consistency.

If failures occur while writing configuration data into the backend data store, the `com.sun.star.configuration.ConfigurationProvider` resynchronizes with the data stored in the backend. The listeners are notified of any differences as if they had been stored through another view. If an application has more stringent error handling needs, the caching behavior can be adjusted by providing arguments when creating the view.

In summary, , there are few overall guarantees regarding transactional integrity for the configuration database, but locally, the configuration behaves as if the support is in place. Depending on the backend capabilities, the `com.sun.star.configuration.ConfigurationProvider` tries to provide the best approximation to transactional integrity that can be achieved considering the capabilities of the backend without compromising performance.

The following example demonstrates how the configuration interfaces are used to feed a user-interface for preference changes. This shows the framework needed to update configuration values, and demonstrates how listeners are used with configuration views. This example concentrates on properties in group nodes with a fixed structure. It uses the same OpenOffice.org Calc grid settings as the previous example. It assumes that there is a `class GridOptionsEditor` that drives a dialog to display and edit the configuration data:

```
// This method simulates editing configuration data using a GridEditor dialog class
public void editGridOptions() throws com.sun.star.uno.Exception {
    // The path to the root element
    final String cGridOptionsPath = "/org.openoffice.Office.Calc/Grid";

    // create a synchronous view for better error handling (lazywrite = false)
    Object xViewRoot = createUpdatableView(cGridOptionsPath, false);

    // the 'editor'
    GridOptionsEditor dialog = new GridOptionsEditor();

    // set up the initial values and register listeners
    // get a data access interface, to supply the view with a model
    XMultiHierarchicalPropertySet xProperties = (XMultiHierarchicalPropertySet)
        UnoRuntime.queryInterface(XMultiHierarchicalPropertySet.class, xViewRoot);

    dialog.setModel(xProperties);

    // get a listener object (probably an adapter) that notifies
    // the dialog of external changes to its model
    XChangesListener xListener = dialog.createChangesListener( );

    XChangesNotifier xNotifier =
        (XChangesNotifier)UnoRuntime.queryInterface(XChangesNotifier.class, xViewRoot);

    xNotifier.addChangesListener(xListener);

    if (dialog.execute() == GridOptionsEditor.SAVE_SETTINGS) {
        // changes have been applied to the view here
        XChangesBatch xUpdateControl =
            (XChangesBatch) UnoRuntime.queryInterface(XChangesBatch.class,xViewRoot);

        try {
           xUpdateControl.commitChanges();
        }
        catch (Exception e) {
            dialog.informUserOfError( e );
        }
```

```
    }

    // all changes have been handled - clean up and return
    // listener is done now
    xNotifier.removeChangesListener(xListener);

    // we are done with the view - dispose it
    ((XComponent)UnoRuntime.queryInterface(XComponent.class, xViewRoot)).dispose();
}
```

In this example, the dialog controller uses the `com.sun.star.beans.XMultiHierarchicalPropertySet` interface to read and change configuration values. If the grid options are changed and committed in another view, `com.sun.star.util.XChangesListener:changesOccurred()` is sent to the listener supplied by the dialog which can then update its display accordingly.

Note that a synchronous `com.sun.star.configuration.ConfigurationUpdateAccess` was created for this example (argument `lazywrite==false`). As the action here is driven by user interaction, synchronous committing is used to detect errors immediately.

Besides the values for the current user, there are also default values that are determined by merging the schema with any default layers. It is possible to retrieve the default values for individual properties, and to reset a property or a set node to their default states, thus backing out any changes done for the current user. For this purpose, group nodes support the interfaces `com.sun.star.beans.XPropertyState` and `com.sun.star.beans.XMultiPropertyStates`, offering operations to query if a property assumes its default state or the default value, and to reset an updatable property to its default state. The `com.sun.star.beans.PropertyAttribute` structss available through `com.sun.star.beans.XPropertySetInfo:getProperty()` are used to determine if a particular item or node supports this operation.

Individual set elements can not be reset because set nodes do not support `com.sun.star.beans.XPropertyState`. Instead a `com.sun.star.configuration.SetAccess` supports `com.sun.star.beans.XPropertyWithState`that resets the set as a whole.

The following is an example code using this feature to reset the OpenOffice.org Calc grid settings used in the preceding examples to their default state:

```
/// This method resets the grid settings to their default values
protected void resetGridConfiguration() throws com.sun.star.uno.Exception {
    // The path to the root element
    final String cGridOptionsPath = "/org.openoffice.Office.Calc/Grid";

    // create the view
    Object xViewRoot = createUpdatableView(cGridOptionsPath);

    // ### resetting a single nested value ###
    XHierarchicalNameAccess xHierarchicalAccess =
        (XHierarchicalNameAccess)UnoRuntime.queryInterface(XHierarchicalNameAccess.class, xViewRoot);

    // get using absolute name
    Object xOptions = xHierarchicalAccess.getByHierarchicalName(cGridOptionsPath + "/Option");

    XPropertyState xOptionState =
        (XPropertyState)UnoRuntime.queryInterface(XPropertyState.class, xOptions);

    xOptionState.setPropertyToDefault("VisibleGrid");

    // ### resetting more deeply nested values ###
    Object xResolutionX = xHierarchicalAccess.getByHierarchicalName("Resolution/XAxis");
    Object xResolutionY = xHierarchicalAccess.getByHierarchicalName("Resolution/YAxis");

    XPropertyState xResolutionStateX =
        (XPropertyState)UnoRuntime.queryInterface(XPropertyState.class, xResolutionX);
    XPropertyState xResolutionStateY =
        (XPropertyState)UnoRuntime.queryInterface(XPropertyState.class, xResolutionY);

    xResolutionStateX.setPropertyToDefault("Metric");
    xResolutionStateY.setPropertyToDefault("Metric");

    // ### resetting multiple sibling values ###
    Object xSubdivision = xHierarchicalAccess.getByHierarchicalName("Subdivision");

    XMultiPropertyStates xSubdivisionStates =
        (XMultiPropertyStates)UnoRuntime.queryInterface(XMultiPropertyStates.class, xSubdivision);
```

```
    xSubdivisionStates.setAllPropertiesToDefault();

    // commit the changes
    XChangesBatch xUpdateControl =
        (XChangesBatch) UnoRuntime.queryInterface(XChangesBatch.class, xViewRoot);

    xUpdateControl.commitChanges();

    // we are done with the view - dispose it
    ((XComponent)UnoRuntime.queryInterface(XComponent.class, xViewRoot)).dispose();
}
```



Currently, group nodes do not support the attribute `com.sun.star.beans.PropertyAttribute:MAYBEDEFAULT` set in the `com.sun.star.beans.Property` structure available from `com.sun.star.beans.XPropertySetInfo`. Attempts to use `com.sun.star.beans.XPropertyState:setPropertyToDefault` to reset an entire group node fail.

Also, because the group nodes can not be reset, the `com.sun.star.beans.XPropertyState:setPropertyToDefault` or `com.sun.star.beans.XMultiPropertyStates:setAllPropertiesToDefault` cannot be used to reset all descendents of a node.

It is intended to lift this restriction in a future release. To avoid unexpected changes in behavior when this change is introduced, you should apply `com.sun.star.beans.XPropertyState:setPropertyToDefault` only to actual properties, such as value items, or set nodes. In particular, you should avoid `com.sun.star.beans.XMultiPropertyStates:setAllPropertiesToDefault()` on group nodes.

A more comprehensive example is provided that shows how set elements are created and added, and how it employs advanced techniques for reducing the amount of data that needs to be loaded.

This example uses the OpenOffice.org configuration module `org.openoffice.Office.DataAccess`. This component has a set item `DataSources` that contains group items described by the template `DataSourceDescription`. A data source description holds information about the settings required to connect to a data source.

The template `org.openoffice.Office.DataAccess/DataSourceDescription` has the following properties that describe the data source connection:

Name	Type	Comment
`URL`	`String`	Data source URL.
`IsPasswordRequired`	`Boolean`	Is a password needed to connect.
`TableFilter`	`String []`	Filters tables for display.
`TableTypeFilter`	`String []`	Filters tables for display.
`User`	`String`	User name to be used for connecting.
`LoginTimeout`	`int`	Default timeout for connection attempt.
`SuppressVersionColumns`	`Boolean`	Controls display of certain data.
`DataSourceSettings`	`set node`	Contains `DataSourceSetting` entriesthat contain driver-specific settings.
`Bookmarks`	`set node`	Contains `Bookmark` entries that link to related documents, for example, Forms.

It also contains the binary properties `NumberFormatSettings` and `LayoutInformation` that store information for layout and display of the data source contents.It also contains the set items `Tables` and `Queries` containing the layout information for the data access views.

The example shows a procedure that creates and saves basic settings for connecting to a new data source. It uses an asynchronous `com.sun.star.configuration.ConfigurationUpdateAccess`. Thus, when `com.sun.star.util.XChangesBatch:commitChanges` is called, the data becomes

visible at the `com.sun.star.configuration.ConfigurationProvider`, but is only stored in the provider's cache. It is written to the data store at later when the cache is automatically flushed by the `com.sun.star.configuration.ConfigurationProvider`. As this is done in the background there is no exception when the write-back fails.



The recommended methodto configure a new data source is to use the `com.sun.star.sdb.DatabaseContext` service as described in *12.2.1 Database Access - Data Sources in OpenOffice.org API - DatabaseContext*. This is a high-level service that ensures that all the settings required to establish a connection are properly set.

Among the parameters of the routine is the name of the data source that must be chosen to uniquely identify the data source from other parameters directly related to the above properties. There also is a parameter to pass a list of entries for the `DataSourceSettings` set.

The resulting routine is: (Config/ConfigExamples.java)

```
// This method stores a data source for given connection data
void storeDataSource(
        String sDataSourceName,
        String sDataSourceURL,
        String sUser,
        boolean bNeedsPassword,
        int nTimeout,
        com.sun.star.beans.NamedValue [] aDriverSettings,
        String [] aTableFilter ) throws com.sun.star.uno.Exception {

    // create the view and get the data source element in a
    // helper method createDataSourceDescription() (see below)
    Object xDataSource = createDataSourceDescription(getProvider(), sDataSourceName);

    // set the values
    XPropertySet xDataSourceProperties = (XPropertySet)UnoRuntime.queryInterface(
        XPropertySet.class, xDataSource);

    xDataSourceProperties.setPropertyValue("URL", sDataSourceURL);
    xDataSourceProperties.setPropertyValue("User", sUser);
    xDataSourceProperties.setPropertyValue("IsPasswordRequired", new Boolean(bNeedsPassword));
    xDataSourceProperties.setPropertyValue("LoginTimeout", new Integer(nTimeout));

    if (aTableFilter != null)
        xDataSourceProperties.setPropertyValue("TableFilter", aTableFilter);

    // ### store the driver-specific settings ###
    if (aDriverSettings != null) {
        Object xSettingsSet = xDataSourceProperties.getPropertyValue("DataSourceSettings");

        // helper for storing (see below)
        storeSettings( xSettingsSet, aDriverSettings);
    }

    // ### save the data and dispose the view ###
    // recover the view root (helper method)
    Object xViewRoot = getViewRoot(xDataSource);

    // commit the changes
    XChangesBatch xUpdateControl = (XChangesBatch) UnoRuntime.queryInterface(
        XChangesBatch.class, xViewRoot);

    xUpdateControl.commitChanges();

    // now clean up
    ((XComponent) UnoRuntime.queryInterface(XComponent.class, xViewRoot)).dispose();
}
```

Notice the function `createDataSourceDescription` in our example.It is called to get a `DataSourceDescription` instance to access a pre-existing item, or create and insert a new item using the passed name.

The function is optimized to reduce the view to as little data as necessary. To this end it employs the `depth` parameter when creating the view.



The "depth" parameter for optimization purposes is used here for demonstration purposes only. Use of the "depth" flag does not have a noticeable effect on performance with the current implementation of the OpenOffice.org configuration management components. Actually, there are few cases where the use of this parameter has any value.

This results in a view where descendents of the root are only included in the view up to the given nesting depth. In this case, where `depth = 1`, only the immediate children are loaded. If the requested item is found, the function gets a deeper view for only that item, otherwise it creates a new instance. In the latter case, the item returned is not the root of the view. (Config/ConfigExamples.java)

```
/** This method gets the DataSourceDescription for a data source.
    It either gets the existing entry or creates a new instance.

    The method attempts to keep the view used as small as possible. In particular there
    is no view created, that contains data for all data source that are registered.
*/
Object createDataSourceDescription(XMultiServiceFactory xProvider, String sDataSourceName)
        throws com.sun.star.uno.Exception {
    // The service name: Need an update access:
    final String cUpdatableView = "com.sun.star.configuration.ConfigurationUpdateAccess";

     // The path to the DataSources set node
    final String cDataSourcesPath = "/org.openoffice.Office.DataAccess/DataSources";

   // creation arguments: nodepath
    com.sun.star.beans.PropertyValue aPathArgument = new com.sun.star.beans.PropertyValue();
    aPathArgument.Name = "nodepath";
    aPathArgument.Value = cDataSourcesPath ;

    // creation arguments: commit mode
    com.sun.star.beans.PropertyValue aModeArgument = new com.sun.star.beans.PropertyValue();
    aModeArgument.Name = "lazywrite";
    aModeArgument.Value = new Boolean(true);

    // creation arguments: depth
    com.sun.star.beans.PropertyValue aDepthArgument = new com.sun.star.beans.PropertyValue();
    aDepthArgument.Name = "depth";
    aDepthArgument.Value = new Integer(1);

    Object[] aArguments = new Object[3];
    aArguments[0] = aPathArgument;
    aArguments[1] = aModeArgument;
    aArguments[2] = aDepthArgument;

    // create the view: asynchronously updatable, with depth 1
    Object xViewRoot =
        xProvider.createInstanceWithArguments(cUpdatableView, aArguments);

    XNameAccess xSetOfDataSources = (XNameAccess) UnoRuntime.queryInterface(
            XNameAccess.class,xViewRoot);

    Object xDataSourceDescriptor = null; // the result
    if (xSetOfDataSources .hasByName(sDataSourceName)) {
        // the element is there, but it is loaded only with depth zero !
        try {
            // the view should point to the element directly, so we need to extend the path
            XHierarchicalName xComposePath = (XHierarchicalName) UnoRuntime.queryInterface(
                XHierarchicalName.class, xSetOfDataSources );

            String sElementPath = xComposePath.composeHierarchicalName( sDataSourceName );

            // use the name of the element now
            aPathArgument.Value = sElementPath;

            // create another view now (without depth limit)
            Object[] aDeepArguments = new Object[2];
            aDeepArguments[0] = aPathArgument;
            aDeepArguments[1] = aModeArgument;

            // create the view: asynchronously updatable, with unlimited depth
            xDataSourceDescriptor  =
                  xProvider.createInstanceWithArguments(cUpdatableView, aDeepArguments);

            if ( xDataSourceDescriptor != null) // all went fine
            {
                // dispose the other view
                ((XComponent)UnoRuntime.queryInterface(XComponent.class, xViewRoot)).dispose();
                xViewRoot = null;
            }
```

```
        }
        catch (Exception e) {
            // something went wrong, we retry with a new element
            System.out.println("WARNING: An exception occurred while creating a view" +
                " for an existing data source: " + e);
            xDataSourceDescriptor  = null;
        }
    }

    // do we have a result element yet ?
    if (xDataSourceDescriptor == null) {
        // get the container
        XNameContainer xSetUpdate = (XNameContainer)UnoRuntime.queryInterface(
            XNameContainer.class, xViewRoot);

        // create a new detached set element (instance of DataSourceDescription)
        XSingleServiceFactory xElementFactory = (XSingleServiceFactory)UnoRuntime.queryInterface(
            XSingleServiceFactory.class, xSetUpdate);

        // the new element is the result !
        xDataSourceDescriptor  = xElementFactory.createInstance();

        // insert it - this also names the element
        xSetUpdate.insertByName( sDataSourceName ,  xDataSourceDescriptor  );
    }

    return xDataSourceDescriptor ;
}
```

A method is required to recover the view root from an element object, because it is unknown if the item is the root of the view or a descendant : (Config/ConfigExamples.java)

```
// This method get the view root node given an interface to any node in the view
public static Object getViewRoot(Object xElement) {
    Object xResult = xElement;

    // set the result to its parent until that would be null
    Object xParent;
    do {
        XChild xParentAccess =
            (XChild) UnoRuntime.queryInterface(XChild.class,xResult);

        if (xParentAccess != null)
            xParent = xParentAccess.getParent();
        else
            xParent = null;

        if (xParent != null)
            xResult = xParent;
     }
     while (xParent != null);

     return xResult;
}
```

Another function used is `storeDataSource` is `storeSettings` to store an array of `com.sun.star.beans.NamedValues` in a set of `DataSourceSetting` items. A `DataSourceSetting` contains a single property named `Value` tht is set to any of the basic types supported for configuration values. This example demonstrates the two steps required to add a new item to a set node: (Config/ConfigExamples.java)

```
/// this method stores a number of settings in a set node containing DataSourceSetting objects
void storeSettings(Object xSettingsSet, com.sun.star.beans.NamedValue [] aSettings)
        throws com.sun.star.uno.Exception {

    if (aSettings == null)
        return;

    // get the settings set as a container
    XNameContainer xSettingsContainer =
        (XNameContainer) UnoRuntime.queryInterface( XNameContainer.class, xSettingsSet);

    // and get a factory interface for creating the entries
    XSingleServiceFactory xSettingsFactory =
        (XSingleServiceFactory) UnoRuntime.queryInterface(XSingleServiceFactory.class, xSettingsSet);

    // now insert the individual settings
    for (int i = 0; i < aSettings.length; ++i) {
        // create a DataSourceSetting object
        XPropertySet xSetting = (XPropertySet)
            UnoRuntime.queryInterface(XPropertySet.class, xSettingsFactory.createInstance());
```

```
        // can set the value before inserting
        xSetting.setPropertyValue("Value", aSettings[i].Value);

        // and now insert or replace as appropriate
        if (xSettingsContainer.hasByName(aSettings[i].Name))
            xSettingsContainer.replaceByName(aSettings[i].Name, xSetting);
        else
            xSettingsContainer.insertByName(aSettings[i].Name, xSetting);
    }
}
```

Besides adding a freshly created instance of a template, a set item can be removed from a set and added to any other set supporting the same template for its elements, provided both sets are part of the same view. You cannot move a set item between views, as this contradicts the transactional isolation of views. The set item you removed in one view will still be in its old place in the other. If a set item is moved between sets in one view and the changes are committed, the change appears in another overlapping view as removal of the original item and insertion of a new element in the target location, not as relocation of an identical element.



The methods `com.sun.star.container.XNamed:setName()` and `com.sun.star.container.XChild:setParent()` are supported by a `com.sun.star.configuration.ConfigurationUpdateAccess` only if it is a `com.sun.star.configuration.SetElement`. They offer another way to move an item within a set or from one set to another set.

In the current release of OpenOffice.org, these methods are not supported correctly. You can achieve the same effect by using a sequence of remove item - insert item.

To rename an item: (Config/ConfigExamples.java)

```
/// Does the same as xNamedItem.setName(sNewName) should do
void renameSetItem(XNamed xNamedItem, String sNewName) throws com.sun.star.uno.Exception {
    XChild xChildItem = (XChild)
        UnoRuntime.queryInterface(XChild.class, xNamedItem);

    XNameContainer xParentSet = (XNameContainer)
        UnoRuntime.queryInterface(XNameContainer.class, xChildItem.getParent());

    String sOldName = xNamedItem.getName();

    // now rename the item
    xParentSet.removeByName(sOldName);
    xParentSet.insertByName(sNewName,xNamedItem);
}
```

To move an item to a different parent: (Config/ConfigExamples.java)

```
/// Does the same as xChildItem.setParent( xNewParent ) should do
void moveSetItem(XChild xChildItem, XNameContainer xNewParent) throws com.sun.star.uno.Exception {
    XNamed xNamedItem = (XNamed)
        UnoRuntime.queryInterface(XNamed.class, xChildItem);

    XNameContainer xOldParent = (XNameContainer)
        UnoRuntime.queryInterface(XNameContainer.class, xChildItem.getParent());

    String sItemName = xNamedItem.getName();

    // now rename the item
    xOldParent.removeByName(sItemName);
    xNewParent.insertByName(sItemName,xChildItem);
}
```

15.5 Customizing Configuration Data

The configuration management API is a data manipulation API. There is no support for data definition functionality. You cannot programmatically inspect, modify or create a configuration schema.

This release does not support adding configuration data for your own components by createing and installing a new configuration data file into a backend manually. The file format used for the current standard backend is not documented, and there is no documentation about the internal organization of the standard file-based backend and proper deployment of selfmade configuration data files. A migration to new, well-documented file formats that are still XML-based and to a documented organization of the data files in the standard backend are being prepared. This feature will become available in a future version of OpenOffice.org. For more information, visit *http://util.openoffice.org*.

15.6 Adding a Backend Data Store

At present, the code to select and access a particular data store is hardcoded into the configuration management component. We are working on providing a UNO-based interface to enable different data stores in a flexible manner. This feature will become available in a future version of OpenOffice.org. For more information, visit *http://util.openoffice.org*.

16 Office Bean

16.1 Introduction

This chapter describes the OfficeBean Java Bean component. It is assumed that the reader is familiar with the Java Beans technology. Additional information about Java Beans can be found at *http://java.sun.com/beans*.

With the OfficeBean, a developer can easily write Java applications, harnessing the power of OpenOffice.org. It encapsulates a connection to a locally running OpenOffice.org process, and hides the complexity of establishing and maintaining that connection from the developer.

It also allows embedding of OpenOffice.org documents within the Java environment. It provides an interface the developer can use to obtain Java AWT windows into which the backend OpenOffice.org process draws its visual representation. These windows are then plugged into the UI hierarchy of the hosting Java application. The embedded document is controlled from the Java environment, since the OfficeBean allows developers to access the complete OpenOffice.org API from their Java environment giving them full control over the embedded document, its appearance and behavior.

The OfficeBean consists of two parts. The *officebean.jar* implements a fundamental framework that is able to connect to the office and display the application window of a local OpenOffice.org installation in a Swing frame. The other part has three example beans that take advantage of the officebean.jar to implement Java beans to display the OpenOffice.org documents. The example beans are the BasicOfficeBean, SimpleBean and OfficeWriter.

The `BasicOfficeBean` is a java.awt.Container that encapsulates office documents. It connects to the office, loads documents, tracks their modified status and saves them.

The `SimpleBean` and `OfficeWriter` are derived from `BasicOfficeBean`. The `SimpleBean` is used to toggle the menu bar. The `OfficeWriter` wraps a writer document, for example, by providing its content as a string and as XText interface, and is an extension of `Office` that is based on the `BasicOfficeBean`. The `Office` enhances `BasicOfficeBean` by handling streams, controlling toolbars and menu bar, publishing the global service manager, and returning the current selection.

The code snippet below shows how to use the OfficeBean API to display a OpenOffice.org document in a Java environment using the example class `SimpleBean`. It creates a SimpleBean, applies an OfficeConnection to it and adds the connected office bean to a frame. When it is added to a frame, SimpleBean loads a OpenOffice.org document from a URL, such as *private:factory/swriter*:

```
public loadDocument(String url) {
    Frame f = new Frame();

    OfficeConnection officeConnection = new LocalOfficeConnection();
    SimpleBean simpleBean = new SimpleBean();

    try {
        simpleBean.setOfficeConnection(officeConnection);
```

```
        f.add(simpleBean, BorderLayout.CENTER);
        simpleBean.load(url);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Illustration 204 shows the resulting OpenOffice.org document window embedded within a java.awt.Frame.

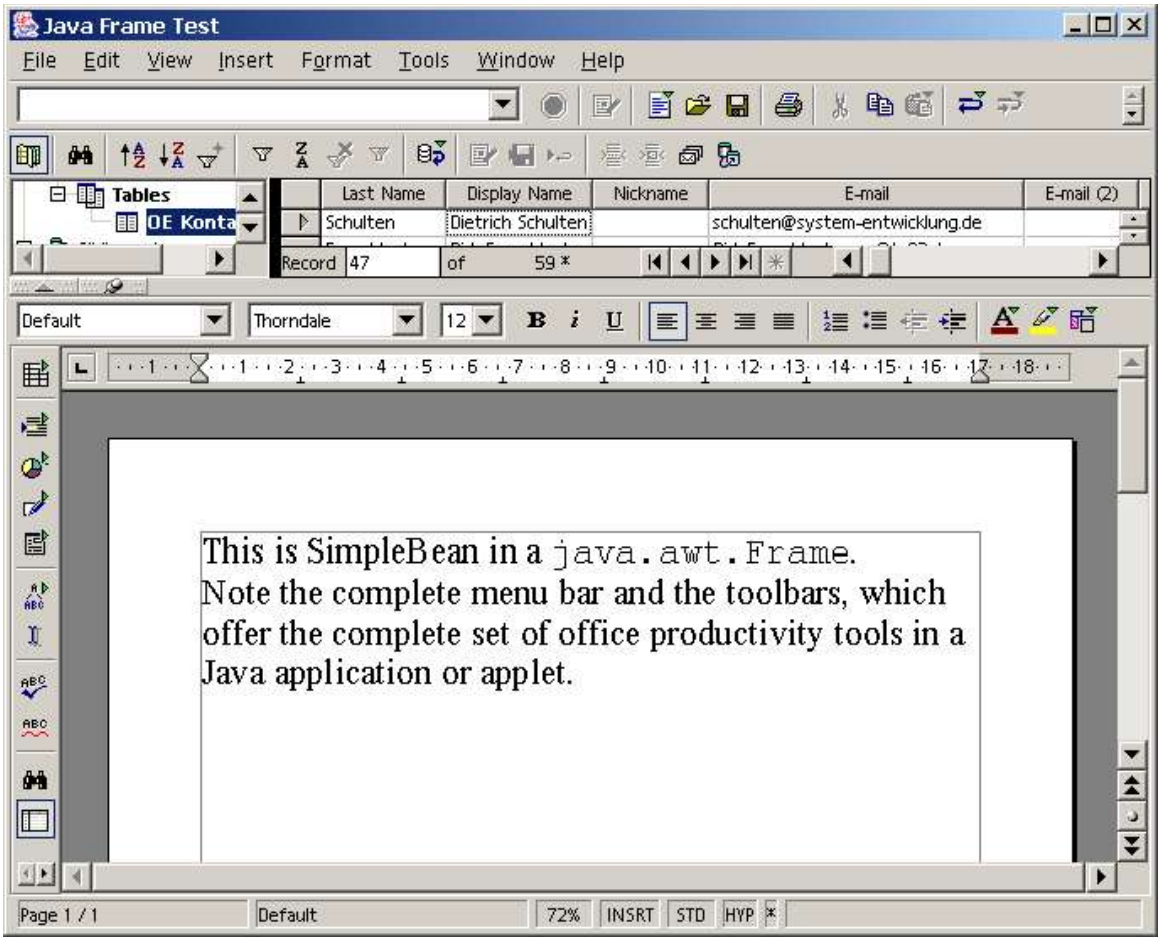


Illustration 204

16.2 Overview of the OfficeBean API

The OfficeBean API is exported in two Java interfaces, `com.sun.star.beans.OfficeConnection` and `com.sun.star.beans.OfficeWindow`.



Note that these interfaces are Java interfaces in the `com.sun.star.beans` package, they are not UNO interfaces.

An implementation of `com.sun.star.beans.OfficeConnection` is provided in the class `com.sun.star.beans.LocalOfficeConnection`. The class com.sun.star.beans.LocalOfficeWindow implements `com.sun.star.beans.OfficeWindow`. The relationship between the `OfficeBean` interfaces and their implementation classes is shown in the illustration below.

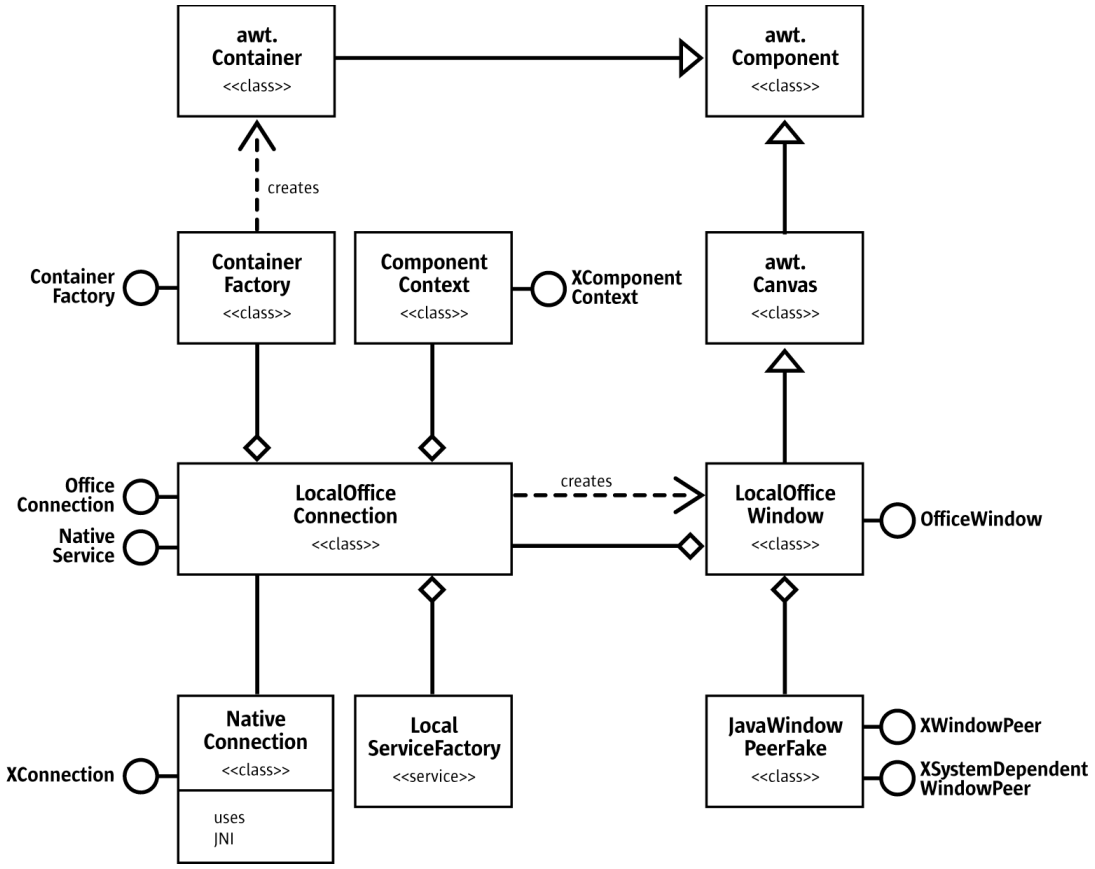


Illustration 205

The following sections describe the OfficeBean interfaces `OfficeConnection` and `OfficeWindow`.Refer to the section "Using the OfficeBean" for an explanation of how the implementation classes are used.

16.2.1 OfficeConnection Interface

The `com.sun.star.beans.OfficeConnection` interface contains the methods used to configure, initiate,and manage the connection to OpenOffice.org. These methods are:

```
public void setUnoUrl(String URL) throws java.net.MalformedURLException
public com.sun.star.uno.XComponentContext getComponentContext()
public OfficeWindow createOfficeWindow(Container container)
public void setContainerFactory(ContainerFactory containerFactory)
```

The client uses `setUnoUrl()` to specify to the OfficeBean how it connects to the OpenOffice.org process. See the section “Configuring the OfficeBean” for a description of the syntax of the URL. A `java.net.MalformedURLException` is thrown by the concrete implementation if the client passes a badly formed URL as an argument.

The method `getComponentContext()` gets an object that implements the `com.sun.star.uno.XComponentContext` interface from the OfficeBean. This object is then used to obtain objects implementing the full OpenOffice.org API from the backend OpenOffice.org process.

A call to `createOfficeWindow()` requests a new `OfficeWindow` from the `OfficeConnection`. The client obtains the `java.awt.Component` from the `OfficeWindow` to plug into its UI. See the

`getAWTComponent()` method below on how to obtain the Component from the `OfficeWindow`. The client provides `java.awt.Container` that indicates to the implementation what kind of `OfficeWindow` it is to create.

The method `setContainerFactory()` specifies to the OfficeBean the factory object it uses to create Java AWT windows to display popup windows in the Java environment. This factory object implements the `com.sun.star.beans.ContainerFactory` interface. See below for a definition of the `ContainerFactory` interface.

If the client does not implement its own `ContainerFactory` interface, the OfficeBean uses its own default `ContainerFactory` creating instances of java.awt.Canvas.

16.2.2 OfficeWindow Interface

The `com.sun.star.beans.OfficeWindow` interface encapsulates the relationship between the AWT window that the client plugs into its UI, and the `com.sun.star.awt.XWindowPeer` object which the OpenOffice.org process uses to draw into the window. It provides two public methods:

```
public java.awt.Component getAWTComponent()
public com.sun.star.awt.XWindowPeer getUNOWindowPeer()
```

The client uses `getAWTComponent()` to obtain the Component window associated with an OfficeWindow. This Component is then added to the clients UI hierarchy.

The method `getUNOWindowPeer()` obtains the UNO `com.sun.star.awt.XWindowPeer` object associated with an `OfficeWindow`.

16.2.3 ContainerFactory Interface

The interface `com.sun.star.beans.ContainerFactory` defines a factory class the client implements if it needs to control how popup windows generated by the backend OpenOffice.org process are presented within the Java environment. The factory has only one method:

```
public java.awt.Container createContainer()
```

It returns a `java.awt.Container`.



For more background on handling popup windows generated by OpenOffice.org, and possible threading issues to consider, see *6.1.7 Office Development - OpenOffice.org Application Environment - Java Window Integration*.

16.3 LocalOfficeConnection and LocalOfficeWindow

The class `LocalOfficeConnection` implements a connection to a locally running OpenOffice.org process that is an implementation of the interface `OfficeConnection`. Its method `createOfficeWindow()` creates an instance of the class `LocalOfficeWindow`, that is an implementation of the interface `OfficeWindow`.

Where `LocalOfficeConnection` keeps a single connection to the OpenOffice.org process, there are multiple, shared `LocalOfficeWindow` instances for multiple beans. The `LocalOfficeWindow` implements the embedding of the local OpenOffice.org document window into a `java.awt.Container`.

16.4 Configuring the OfficeBean

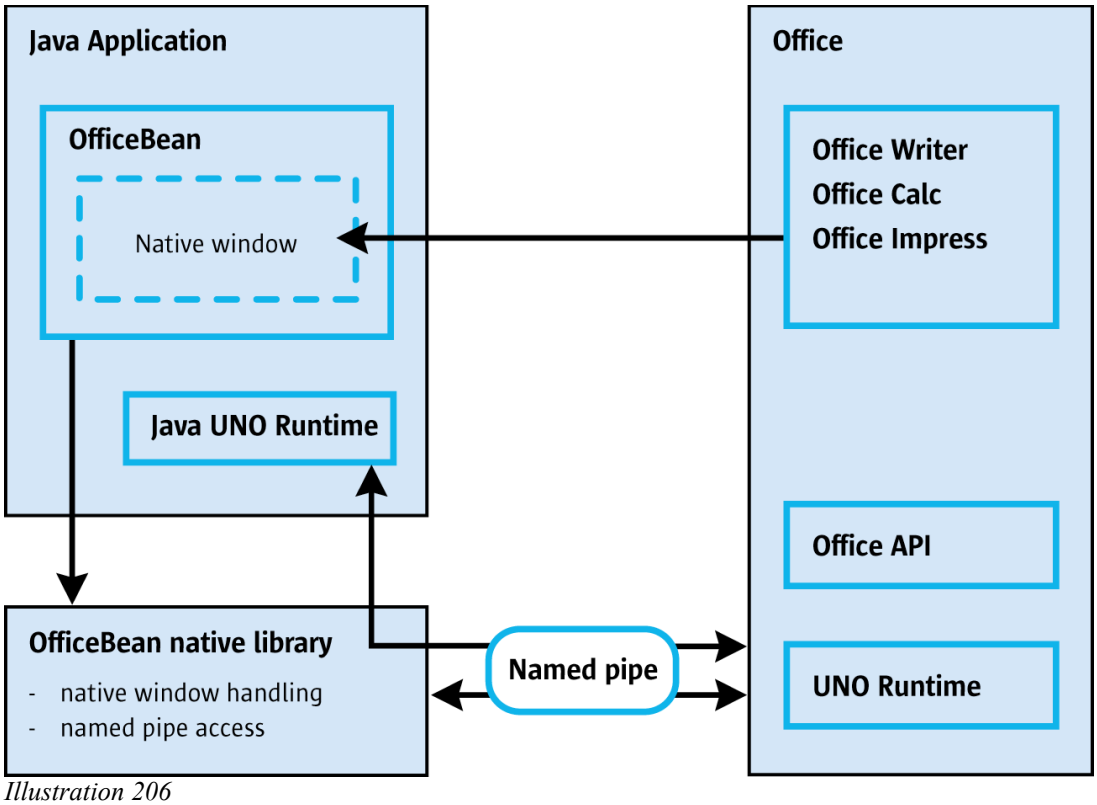


Illustration 206

The fundamental framework of the OfficeBean is contained in the *officebean.jar* archive file that depends on a local library *officebean.dll* or *libofficebean.so*, depending on the platform. The interaction between the backend OpenOffice.org process, officebean local library, OfficeBean and the Java environment is shown in the illustration below.

The OfficeBean allows the developer to connect to and communicate with the OpenOffice.org process through a named pipe. It also starts up a OpenOffice.org instance if it cannot connect to a running office. This is implemented in the OfficeBean local library. The OfficeBean depends on three configuration settings to make this work. It has to find the local library, needs the location of the OpenOffice.org executable, and the bean and office must know the pipe name to use.

16.4.1 Default Configuration

The OfficeBean uses default values for all the configuration settings, if none are provided:

- Since OpenOffice.org 1.1 the *officebean.jar* is located in the *<OfficePath>/program/classes* directory.
- It looks for the local library (Windows: *officebean.dll*, Unix: *libofficebean.so)* relative to the *officebean.jar* in the *<OfficePath>/program* directory. The local library depends on the following shared libraries:

  a) The library sal3 (Windows: *sal3.dll*, Unix: *libsal3.so*) is located in the *<OfficePath>/program* folder. It maybe necessary to add the *<OfficePath>/program* folder to the PATH environment variable if the bean cannot find sal3.

b) The library jawt.dll is needed in Windows. If the bean cannot find it, check the Java Runtime Environment binaries (*<JRE>/bin*) in your PATH environment variable.

- It expects the OpenOffice.org installation in the default install location for the current platform. The *soffice* executable is in the program folder of a standard installation.
- The pipe name is created using the value of the `user.name` Java property. The name of the pipe is created by appending `"_office"` to the name of the currently logged on user, for example, if the `user.name` is `"JohnDoe"`, the name of the pipe is `"JohnDoe_office"`.

Based on these default values, the OfficeBean tries to start the office in listening mode with the -accept commandline option. The exact parameters used by the bean are:

```
# WINDOWS
soffice.exe -bean -accept=pipe,name=<user.name>_Office;urp;StarOffice.NamingService
# UNIX
soffice -bean "-accept=pipe,name=<user.name>_Office;urp;StarOffice.NamingService"
```



There is a limitation in the communication process with the OfficeBean and older versions of OpenOffice.org. If a OpenOffice.org process is already running that was not started with the proper `-accept=pipe` option, the OfficeBean does not connect to it. Since OpenOffice.org 1.1 this limitation is obsolete.

It opens a Writer document outside of the Java frame. The OpenOffice.org process has to be started so that it opens a properly named pipe to enable OpenOffice.org to be displayed as an embedded OfficeBean and top-level OpenOffice.org window. This is achieved by editing the file *<OfficePath>\user\config\registry\instance\org\openoffice\Setup.xml*. Within the <Office/> element, the developer adds an <ooSetupConnectionURL/> element with settings for a named pipe. The following example shows a user-specific *Setup.xml* that configures a named pipe for a user named `JohnDoe`:

```
<?xml version="1.0" encoding="UTF-8"?>
<Setup state="modified" cfg:package="org.openoffice"
  xmlns="http://openoffice.org/2000/registry/components/Setup"
  xmlns:cfg="http://openoffice.org/2000/registry/instance"
  xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance">
 <Office>
  <ooSetupConnectionURL cfg:type="string">
    pipe,name=JohnDoe_Office;urp;StarOffice.NamingService
  </ooSetupConnectionURL>
  <Factories cfg:element-type="Factory">
   <Factory cfg:name="com.sun.star.text.TextDocument">
     <ooSetupFactoryWindowAttributes cfg:type="string">
       193,17,1231,1076;1;
     </ooSetupFactoryWindowAttributes>
   </Factory>
  </Factories>
 </Office>
</Setup>
```

With this user-specific *Setup.xml* file, the office opens a named pipe `JohnDoe_Office` whenever it starts up. It does not matter if the user double clicks a document, runs the Quickstarter, or starts a new, empty document from a OpenOffice.org template.

16.4.2 Customized Configuration

Besides these default values, the OfficeBean is configured to use other parameters. There are three possibilities,using a UNO URL with path and pipe name parameters, setting Java system properties at runtime, or creating a Java property file in the user.home directory.

The first method that a developer uses to configure the OfficeBean is through the UNO URL passed in the `setUnoUrl()` call. The syntax of the UNO URL is as follows:

```
    url    := 'uno:localoffice'[','<params>]';urp;StarOffice.NamingService'
    params := <path>[','<pipe>]
```

```
path   := 'path='<pathv>
pipe   := 'pipe='<pipev>
pathv  := platform_specific_path_to_the_local_office_distribution
pipev  := local_office_connection_pipe_name
```

Here is an example of how to use `setUnoUrl()` in code:

```
OfficeConnection officeConnection = new LocalOfficeConnection();
officeConnection.setUnoUrl(
    "uno:localoffice,path=/home/user/staroffice6.0/program;urp;StarOffice.NamingService");
```



In OpenOffice.org 1.1 the properties mechanism was removed and cannot be used any longer. The following section about the OfficeBean properties and the *officebean.properties file* are only valid for older OpenOffice.org versions. Since OpenOffice.org 1.1 the OfficeBean uses an implicit find mechanism over the classpath for the office and the local OfficeBean library so that no properties file is necessary.

The second method that is used to configure the OfficeBean is using Java system properties. The properties supported by the OfficeBean are:

Properties supported by the OfficeBean	
`com.sun.star.beans.path`	Specifies the path to the *program* directory of the OpenOffice.org installation.
`com.sun.star.beans.libpath`	Specifies the directory the OfficeBean local library (Windows: officebean.dll, Unix: libofficebean.so) is stored The libpath property seems to be ignored, at least on Windows

These properties are set through a call to `System.setProperty()` before creating the `OfficeConnection`, for example:

```
System.setProperty("com.sun.star.beans.path", "/home/user/staroffice6.0/program");
System.setProperty("com.sun.star.beans.libpath", "/home/user/lib");
OfficeConnection officeConnection = new LocalOfficeConnection();
```

The properties are also set by creating a file *.officebean.properties* on Unix or *officebean.properties* with no leading dot on Windows in the directory corresponding to the platform specific `user.home` Java property, that is, the directory returned by `System.getProperty("user.home")`.



In NetBeans, you can look up the home directory in the **Detail** tab of the **Help - About** dialog. Look for the **Home dir** entry.

A possible *officebean.properties* file on a Windows machine may look like the following:

```
com.sun.star.beans.path=D:\\StarOffice6.0\\program
com.sun.star.beans.libpath=X:\\SDK\\windows\\bin
```

On Unix possible *.officebean.properties* could be*:*

```
com.sun.star.beans.path=/home/user/staroffice6.0/program
com.sun.star.beans.libpath=/home/user/lib
```

16.5 Using the OfficeBean

The officebean.jar Java archive file provided as part of the OpenOffice.org Software Development Kit provides an implementation of the OfficeBean API that developers can use to develop applications embedding OpenOffice.org functionality. The implementation is provided in the previously mentioned classes `LocalOfficeConnection` and `LocalOfficeWindow`.

As previously mentioned, the OfficeBean API and implementation is a low-level interface providing the building blocks necessary for developing OfficeBeans. The examples directory of the OpenOffice.org Software Development Kit contains examples of these beans that show how to use these building blocks to build and embed an OfficeBean in a Java applet or application. To use the

low-level API to communicate with the backend OpenOffice.org process, use the `LocalOfficeConnection` and `LocalOfficeWindow` classes directly. Use or extend the beans in the examples to avoid the complexity of using the low-level API.

Basic bean functionality is provided in the abstract class `BasicOfficeBean` that is subclassed to create beans supporting OpenOffice.org document loading and storing from Java. The `BasicOfficeBean` is a `java.awt.Container`, therefore subclasses of `BasicOfficeBean` are plugged directly into a Java UI hierarchy.



Since the OfficeBean uses a native peer to render OpenOffice.org documents, Swing components, such as drop-down menus or list boxes appear behind it, or r they are not displayed at all! To avoid this, exclusively employ AWT components when using the OfficeBean.

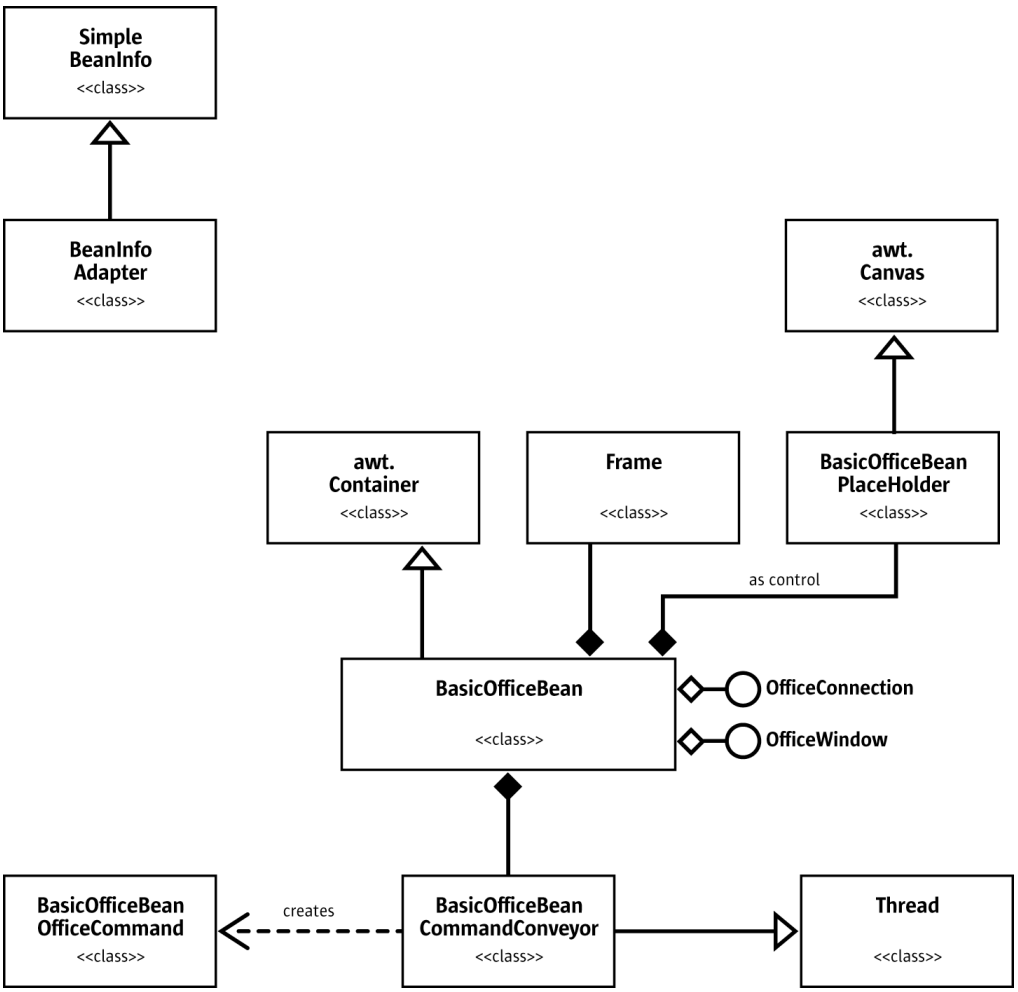


Illustration 207



For readability, the try or catch blocks have been removed from the examples below. The full source code of the examples is found in the OfficeBean directory of the SDK Java examples in *<SDK>/examples/DevelopersGuide/OfficeBean.*

16.5.1 SimpleBean Example

The `SimpleBean` is a concrete subclass of `BasicOfficeBean` used on the component palette of Java IDEs, such as NetBeans (*http://www.netbeans.org*).

Using SimpleBean

To use SimpleBean, build the SimpleBean example from *<SDK>/examples/java/OfficeBean/SimpleBean*.

Installing SimpleBean

Before building and installing the SimpleBean, install and configure the following tools.

Java Development Kit (JDK)
: A JDK from version 1.3.1 is required to build SimpleBean.

OpenOffice.org Software Development Kit (SDK)
: An installation of the OpenOffice.org Software Development Kit (SDK) is required in your development environment. Refer to the SDK documentation for details on how to install the SDK.

GNU make
: The SDK depends on a GNU make version 3.79 or later found at www.gnu.org, and the local windows executables are available from *www.nextgeneration.dk* or *unxutils.sourceforge.net*. Note that on Windows the current *Cygwin make* is known to cause problems with the OpenOffice.org SDK.

After setting up the SDK, start a commandline shell, change to the SDK folder and run the *setsdkenv* script created by *configure*. This script creates a number of environment variables that are required for the make process. Change to the *SimpleBean* directory and execute *make*. The *make* utility runs the *makefile* in the current folder that creates a *simplebean.jar* in the platform-specific *. out* folder of the SDK. The following steps are a possible method to install SimpleBean into the NetBeans IDE.

- Copy *simplebean.jar* from *<SDK>/<Platform>example.out* and *officebean.jar* from *<SDK>/classes* to *<NetBeans>/beans*.
- Run Netbeans. If you want, create a new NetBeans project using **Project – Project Manager**.
- Tell NetBeans where to look for *officebean.jar* and the Java UNO runtime archives, because the SimpleBean needs these files , if we want to add SimpleBean to the component palette. Select **Tools – Options**. Next, open the node **IDE Configuration – System – FileSystems Settings** and right-click **FileSystems Settings**. From the context menu, choose **New – Archive (JAR, Zip)** and select *officebean.jar* from *<NetBeans>/beans*. Use the same context menu entry to add all jar files from *<OfficePath>/program/classes*. Close the **Options** window.
- Choose **Tools – Install New JavaBean**, navigate to *simplebean.jar* and press **OK** to import. NetBeans prompts you to select the only available bean in the archive (SimpleBean) and asks you to determine the palette category where SimpleBean should appear. Choose the category **Beans**. An **OfficeBean** icon is displayed in the **Beans** category of the component palette, which stands for SimpleBean.



Illustration 208

Putting SimpleBean to Work

The following example applet `SimpleViewer` demonstrates the usage of `SimpleBean`. It employs an instance of `SimpleBean` to load a OpenOffice.org document. (OfficeBean/SimpleBean/SimpleViewer.java)

```
public class SimpleViewer extends java.applet.Applet {
    private OfficeConnection localOfficeConnection;
    private SimpleBean simpleBean;

    public void init() {
        setLayout(new BorderLayout());

        // initialize the rest of the Java GUI including a button to create a blank document
        // the method createNewDocument will be called when it is clicked

        // create a SimpleBean and add it to our applet
        simpleBean = new SimpleBean();
        add(simpleBean, BorderLayout.CENTER);
    }

    // load a document
    public void createNewDocument(String url) {
        // if there is no connection to an Office process we need to connect to one
        if (localOfficeConnection == null)
            localofficeConnection = new LocalOfficeConnection();

        // tell the bean to use localOfficeConnection
        simpleBean.setOfficeConnection(localofficeConnection);
        // ask the bean to load the file in url
        simpleBean.load(url);
    }
}
```

In Netbeans, create a new `java.awt.Frame`, drop SimpleBean in, and edit the `Frame` constructor to initialize SimpleBean.

- If a new project is started, create and mount a new local folder for your project files using the context menu of the **FileSystems** node in the NetBeans **Explorer** window. Choose **Mount – Local Directory**.
- Right click the new project folder and click **New – GUI Forms – AWT Forms – Frame**. This template creates a new java.awt.Frame in your project. NetBeans prompts for a name. Enter `SimpleBean1`. Click **Finish** to skip the remaining steps of the **New Frame** wizard.
- In the Netbeans **Explorer,** double-click the new frame **SimpleBean1** that appears in your project folder. The **Form Editor** window pops up and displays an empty frame. Select the tab **Beans** on the component palette, click the OfficeBean icon and drag a rectangle in the middle of the empty frame. This step adds the SimpleBean to the center of the BorderLayout.

  NetBeans throws an exception if you add the current version of SimpleBean1 to a `java.awt.Panel` instead of a `java.awt.Frame`.
- Right click **SimpleBean1** in the **Form Editor** and choose **Goto Source** to display the constructor of `SimpleBean1`. Enter the following code after the generated call to `initComponents()`. This loads a blank document into the frame upon frame creation.

```
public class SimpleBean1 extends java.awt.Frame {

    /** Creates new form SimpleBean1 */
```

```
    public SimpleBean1() {
        initComponents();

        // create Connection
        com.sun.star.beans.OfficeConnection connection1 =
            new com.sun.star.beans.LocalOfficeConnection();

        // set up simpleBean1 with connection1
        simpleBean1.setOfficeConnection(connection1);

        // load blank document
        try {
            simpleBean1.load("private:factory/swriter");
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    ...
}
```

- Click **Debug – Start** to test and debug SimpleBean1. You are now ready to set up an AWT toolbar with buttons to load and create documents, as required.

SimpleBean Internals

Let us look at how `BasicOfficeBean` loads a document to understand how a bean uses the OfficeBean API. The `BasicOfficeBean.load()` method checks if an `OfficeWindow` has been created and calls the `BasicOfficeBean.openConnection()` method to create one, if required:

```
public synchronized void load(String url) throws java.io.IOException {
    try {
        if (mWindow == null)
                openConnection();

    // we consider the complete load() method later
    ...
```

The `BasicOfficeBean.openConnection()` uses the `com.sun.star.beans.OfficeConnection` passed in the `setOfficeConnection()` call, that is, an instance of `com.sun.star.beans.LocalOfficeConnection`, to create an `OfficeWindow`. An instance of the `com.sun.star.lang.XMultiServiceFactory` interface is also obtained through the `OfficeConnection` and stored in the `BasicBean` instance: (OfficeBean/BasicOfficeBean.java)

```
public synchronized void openConnection() throws com.sun.star.uno.Exception {
    if (mWindow != null)
        return;

    // Obtain the global MultiServiceFactory and store it in mServiceFactory
    XMultiComponentFactory  compfactory;
    compfactory  = mConnection.getComponentContext().getServiceManager();

    mServiceFactory  = (XMultiServiceFactory)UnoRuntime.queryInterface(
        XMultiServiceFactory.class, compfactory);

    // Create the OfficeWindow
    mWindow  = mConnection.createOfficeWindow(this);

    // Create the office document frame and initialize the bean
    initialize();
}
```

The `BasicOfficeBean.initialize()` is the last call in `openConnection()` that sets up the objects necessary for document loading: An instance of `com.sun.star.frame.Frame` is initialized with the UNO window peer, and the interfaces `com.sun.star.lang.XMultiServiceFactory` and `com.sun.star.document.XTypeDetection` of a `com.sun.star.frame.FrameLoaderFactory`. (OfficeBean/BasicOfficeBean.java)

```
private void initialize() {
    // Get XWindow interface of UNO window peer
    XWindow window = (XWindow)UnoRuntime.queryInterface(XWindow.class, mWindow.getUNOWindowPeer());
```

```
    // instantiate UNO frame, store its XFrame interface in mFrame
    object = mServiceFactory.createInstance("com.sun.star.frame.Frame");
    mFrame = (Xframe)UnoRuntime.queryInterface(XFrame.class, object);

    // configure the frame to use the UNO window peer
    mFrame.initialize(window);

    // instantiate frame loader factory
    object = mServiceFactory.createInstance("com.sun.star.frame.FrameLoaderFactory");

    // store its XMultiServiceFactory interface in mFrameLoaderFactory
    mFrameLoaderFactory = (XMultiServiceFactory)UnoRuntime.queryInterface(
        XMultiServiceFactory.class, object);

    // store its XTypeDetection interface in mTypeDetector
    mTypeDetector = (XTypeDetection)UnoRuntime.queryInterface(XTypeDetection.class, object);
}
```

The `BasicOfficeBean.load()` method then completes the loading of the document. Instead of `loadComponentFromURL()`, a `com.sun.star.frame.FrameLoader` is employed that expects a frame to load the document into. The following snippet shows the complete `load()` method: (OfficeBean/BasicOfficeBean.java)

```
public synchronized void load(String url) throws java.io.IOException {
    try {
        if (mWindow == null)
            openConnection();

        // Make sure the URL contains something meaningful
        if (url.equals(""))
            url = getDefaultDocumentURL();

        // Find out the type of the document.
        String type = mTypeDetector.queryTypeByURL(url);

        // Get frame loader factory for document type
        Object object = mFrameLoaderFactory.createInstance(type);
        XSynchronousFrameLoader frameLoader;
        frameLoader = (XSynchronousFrameLoader)UnoRuntime.queryInterface(
            XSynchronousFrameLoader.class, object);

        // Create the document descriptor with two PropertyValue structs:
        // FileName = url and TypeName = type
        PropertyValue[] desc = new PropertyValue[2];
        desc[0] = new PropertyValue("FileName", 0, url, PropertyState.DIRECT_VALUE);
        desc[1] = new PropertyValue("TypeName", 0, type, PropertyState.DIRECT_VALUE);

        // Avoid Dialog 'Document changed' while reloading
        try {
            setModified(false);
        } catch (java.lang.IllegalStateException exp) {
        }

        // Load the document and store the URL
        if (frameLoader.load(desc, mFrame) == false) {
            throw new java.io.IOException("Can not load a document: \"" + url + "\"");
        }
        mDocumentURL = url;

        // Get document's XModifiable interface if any and store it
        if ((mFrame != null) && (mFrame.getController() != null)) {
            XModel model = mFrame.getController().getModel();
            mModifiable = (Xmodifiable)UnoRuntime.queryInterface(XModifiable.class, model);
        }
        else {
            mModifiable = null;
        }

        // Find top most parent and force it to validate.
        Container parent = this;
        while (parent.getParent() != null)
            parent = parent.getParent();
        ((Window)parent).validate();
    }
    catch (com.sun.star.uno.Exception exp) {
        throw new java.io.IOException(exp.getMessage());
    }
}
```

16.5.2 OfficeWriterBean Example

The `DocViewer` is a Java application that loads a new or opens an existing Writer document, and prints the selected text into a text field. It updates the text field as the selected text changes. The `DocViewer` depends on two bean classes to display and manipulate the document: `Office` and `OfficeWriter`.

The `Office` and `OfficeWriter` are concrete classes that show how the `BasicOfficeBean` is extended to allow the developer to manipulate an embedded document programmatically. The `Office` shows how to add the ability to execute commands on the backend OpenOffice.org process using the dispatch framework. Refer to chapter *6 Office Development*). It uses instances of the `OfficeCommand` class, also provided with the examples, to represent OpenOffice.org command URLs that are applied to the document. The `Office.setMenuBarVisible()` is an example of how to do this: (OfficeBean/OfficeWriterBean/Office.java)

```
public void setMenuBarVisible(boolean visible) {
    if (mMenuBarVisible != visible) {
        if (isDocumentLoaded() == true) {
            OfficeCommand command = new OfficeCommand(SID_TOGGLEMENUBAR);
            command.appendParameter("MenuBar", new Boolean(visible));
            command.execute(this);
        }
        mMenuBarVisible  = visible;
        firePropertyChange("MenuBarVisible", new Boolean(mMenuBarVisible), new Boolean(visible));
    }
}
```

`OfficeWriter` extends `Office` further to allow the developer to get and set the contents of the document, obtain the `com.sun.star.text.XTextDocument` interface for the document, get the selected text in the document, and listen for changes to the document. The code below shows how `OfficeWriter` is added to the `DocViewer` UI, and how `DocViewer` listens for changes to the document loaded by `OfficeWriter`: (OfficeBean/OfficeWriterBean/DocViewer.java)

```
public class DocViewer {
    public void initUI() {
        Frame frame = new Frame();
        JTextField currentSelection = new JTextField();

        // set up the rest of the application UI

        mOfficeWriter = new OfficeWriter();
        mOfficeWriter.setOfficeConnection(new LocalOfficeConnection());
        mOfficeWriter.addSelectionChangeListener(new DocViewerChangeListener());
        frame.add(mOfficeWriter, BorderLayout.CENTER);
    }
```

The following shows the `ChangeListener` that DocViewer uses to listen for changes in the document selection:(OfficeBean/OfficeWriterBean/DocViewer.java)

```
    class DocViewerChangeListener implements ChangeListener {
        public void stateChanged(ChangeEvent event)
        {
            String s = mOfficeWriter.getSelection();
            currentSelection.setText(s);
        }
    }
```

The examples above provide an overview of how the OfficeBean API is used to create Java beans that can be used in Java applications and applets. BeanInfo classes are provided for the `SimpleBean`, `Office` and `OfficeWriter` for integrating within an IDE (Integrated Development Environment), such as the Bean Development Kit or Forte for Java. Developers can use the examples as a guideline when using the OfficeBean API to write new beans, or use or extend the example beans.

17 Accessibility

There are certain circumstances where OpenOffice.org applications can not be used with the usual input and output devices, such as a mouse, keyboard, monitor and printer. This may be because the user is sitting in a car and can only occasionally look at the screen and has no keyboard at all. Maybe the user is disabled and can not see or hear or use traditional keyboards. Alternative input and output devices are called *assistive technology*, or AT. Examples of AT are Braille terminals, which are used mainly for display of single text lines where each character is represented by raised or lowered dots and can be read by touching them with the finger tips, screen magnifiers, which magnify the screen contents and optionally change color, and screen readers, which use speech synthesis to read displayed text or descriptions of objects out loud in a human language.

To make OpenOffice.org applications accessible to the disabled or to people in mobile environments, alternative input and output devices have to be supported. In order to support a wide variety of ATs, the approach taken by Java and Gnome has been adopted: an API tailored to the specific needs of AT and modeled closely after its Java counterpart is used as an interface between the available data of the elements visible on screen and the AT, which transforms that data and presents an alternative view of the screen contents.

17.1 Overview

As previously stated, the UNO Accessibility API, or UAA, is closely modeled after the Java Accessibility API, and to some extent the Gnome Accessibility API. This section describes some differences with common UNO styles and standards.

The purpose of the accessibility API is to represent what is currently visible on screen. To be kept up-to-date, users of the accessibility API are informed of any relevant changes of the screen content by events. This focus on visual appearance is another point in which the accessibility API differs from other parts of UNO, which are strictly model centered. The accessibility API provides a tree structure or, to be more specific, a forest of accessibility objects that, as a whole, represent the on-screen data.

The transition point from the UNO API to the accessibility API is windows, which both support the `com.sun.star.awt.XWindow` and `com.sun.star.accessibility.XAccessible` interfaces. A list of all top-level windows, from which you can get the roots of the associated accessibility trees, can be retrieved from the toolkit through the `com.sun.star.awt.XExtendedToolkit` interface.

The `com.sun.star.accessibility.XAccessible` interface can be queried for the actual accessibility object with its only function `com.sun.star.accessibility.XAccessible:getAccessible()`. With this technique, the implementations of the accessibility interfaces can be kept apart from that of the other UNO interfaces. The `getAccessible()` function returns a reference to an object that implements the `com.sun.star.accessibility.XAccessibleContext` interface. This interface is the center of the accessibility API. On the one hand it provides the functionality to traverse the accessibility tree and on the other hand gives access to basic information that represents the

object that is being made accessible. These two aspects are described in more detail in the following sections [ref:Accessibility Tree] and [ref:Content Information].

The accessibility API is as self-contained as possible. You should not need to use the UNO API outside its accessibility API subset. However, there are some exceptions to this. The most important one is the initial access to the root nodes of the accessibility tree over the toolkit.

17.2 Bridges

There are several ways that ATs can be realized. They differ in two important points. The first one is how the AT represents the information it obtains from OpenOffice.org and presents it to the user. The second difference is how ATs obtain this information in the first place.

In its simplest form, the communication between AT and OpenOffice.org involves only the accessibility API and, where necessary, some additional features from other parts of the UNO API. Existing ATs, however, do not know anything yet about the accessibility API. They use one of several ways to access OpenOffice.org by using one or more bridges that translate between different APIs:

- The *UNO access bridge* translates between the accessibility API and the Java Accessibility API. Note that this is not the same as the Java version of the accessibility API.
- The *Windows/Java access bridge* translates between the Java and the C versions of the Java Accessibility API.
- The *Gnome access bridge* translates between the accessibility API and the Gnome Accessibility API.

In order to make OpenOffice.org accessible, it is necessary to support the accessibility API. The characteristics of the bridges have to be taken into account as well.

Under Windows, OpenOffice.org itself has a switch that can turn on or off the accessibility support. This switch can be reached through **Tools – Options – Accessibility**. Under Linux and Solaris, an equivalent setting can be made in the Gnome environment. When accessibility is activated, on every launch OpenOffice.org will start its own Java VM, which in turn starts all registered AT tools. This will be explained in the next section.

17.3 Accessibility Tree

The screen content is presented to AT as a tree—or a forest, to be more specific—of accessibility objects. Each displayed object that wants to be accessible has to support the `com.sun.star.accessibility.XAccessible` interface. From this interface, you obtain the actual *accessibility object* by calling the `getAccessibleContext()` function. The returned object has to at least support the `com.sun.star.accessibility.XAccessibleContext` interface.

Accessibility objects are organized in one or more hierarchies, one for each top-level window. So there is a tree for a single top-level window, and a forest when there is more than one top-level window. Internal nodes of a tree are containers of other accessibility objects. A container can represent window frames, toolbars, menus, group shapes, or shapes that contain text. Leaves represent objects like menu entries without sub-menus, buttons, icons, shapes without text, or text paragraphs.

You can move up and down within the tree of a given accessibility object. All functions for obtaining an object's parent and children are part of the

com.sun.star.accessibility.XAccessibleContext interface. The ability to move up towards the tree root is provided by the getAccessibleParent() function. Like all other accessibility functions that return a reference to another accessibility object, it returns a reference to a com.sun.star.accessibility.XAccessible object. Moving down the tree towards the leaves requires two functions. The getAccessibleChildCount() function returns the number of children. The getAccessibleChild() function allows you to access any child by specifying the appropriate index.



Between the call to getAccessibleChildCount() and the final getAccessibleChild() call (when accessing all children one after the other) the number of children may have changed. You can keep track of the number of children by registering as listener and waiting for com.sun.star.accessibility.AccessibleEventId:CHILD events. Additionally, you have to cope with com.sun.star.lang.IndexOutOfBoundsException exceptions that denote bad indices.

When children are added or removed from an accessibility object, the indices of the new and remaining children may change. You can use the getIndexInParent() function to get the current indices.

17.4 Content Information

Content information of accessibility objects establishes the connection to the objects that are visible on the screen. This information gives a detailed description of what is visible on the screen, the size of the object, and its location on the screen. Access to the content information is provided by several interfaces of the UNO accessibility API, which are described in the following sections.

The accessibility API allows you to divide the implementation of an accessible object into an accessibility related and an accessibility unrelated part. This is done with the com.sun.star.accessibility.XAccessible interface. The getAccessibleContext() method returns an object that implements the other interfaces related to accessibility. This object may be the same as the object that is made accessible, but it can be a different object as well. Once you have the accessibility object, you can use the usual UNO type cast mechanisms to change from one interface to another.

17.5 Listeners and Broadcasters

The com.sun.star.accessibility.XAccessibleEventBroadcaster and com.sun.star.accessibility.XAccessibleEventListener interface combo lets you register and unregister listeners. Events are represented by com.sun.star.accessibility.AccessibleEventObject structure. The different event types are listed and explained in the com.sun.star.accessibility.AccessibleEventId constants group.

Again, event types can be divided into two classes depending on whether they describe changes in the structure of the accessibility tree or changes in the internal state or the visual appearance of an accessible object. The first group consists of CHILD and INVALIDATE_CHILDREN. The first denotes a newly inserted or a removed child. The second is used in cases where more than one child has been inserted or removed and tells the listener to re-fetch a complete list of children.

The second group comprises all other event types. Typical members are VISIBLE_DATA_CHANGED and STATE_CHANGED, which inform listeners that the visual appearance of an object has changed (for example, to a different text color) or that one of its states has been switched on or off (for example, when an object becomes focused or selected).

The event types `CONTROLLED_BY_RELATION_CHANGED`, `CONTROLLER_FOR_RELATION_CHANGED`, `LABEL_FOR_RELATION_CHANGED`, `LABELED_BY_RELATION_CHANGED`, `MEMBER_OF_RELATION_CHANGED`, `CONTENT_FLOWS_FROM_RELATION_CHANGED` and `CONTENT_FLOWS_TO_RELATION_CHANGED` may be thought of as constituting a third group. They describe changes of the more virtual structure formed by relations between accessible objects in different parts of an accessibility tree.

Events are sent *after* the respective change of an accessible object took place. This enables the listener to retrieve up-to-date values that are sent with the event.



A problem arises when the number of children becomes very large, as with Calc tables where the number of cells is 256·32000=8192000. Registering at every cell certainly is not an option. The solution to this problem is the introduction of the `com.sun.star.accessibility.AccessibleStateType:TRANSIENT` state, which tells an AT not to register but to expect `com.sun.star.accessibility.AccessibleEventId:ACTIVE_DESCENDANT_CHANGED` events sent from their parent. To prevent the AT from having to ask every child whether it is transient, the parent must set the `com.sun.star.accessibility.AccessibleStateType:MANAGES_DESCENDANTS` state.

17.6 Implementing Accessible Objects

17.6.1 Implementation Rules

There are some rules to observe when implementing the UNO accessibility API that go beyond simply following the specifications in the IDL files of the accessibility API's interfaces. These rules have to do with what kind of data ATs expect from an application.

One such rule is that only objects that are visible on the screen are included into the accessibility tree. If, for example, you have a text document with a large number of pages, usually only parts of one or two pages are visible, and only accessibility objects for these parts should be part of the accessibility hierarchy. Another closely related rule is that the bounding boxes of objects are clipped to the visible area.

However, there are exceptions to these rules. For reasons of consistency with the behavior of Java tables represented through the `com.sun.star.accessibility.XAccessibleTable` interface, access is granted to all of their cells regardless of whether they are visible or not. Menus are another example. The whole menu structure is represented, even when only the menu bar is visible.

Another rule is that bounding boxes of accessibility objects as returned by `com.sun.star.accessibility.XAccessibleComponent:getBounds()` must not overlap the bounding boxes of their parents. This is crucial to enable ATs to find the accessibility object that lies under a given screen coordinate, such as the mouse position. With properly nesting bounding boxes, ATs can prune whole sub-trees from the search when the bounding box of the root object does not contain the point.

17.6.2 Services

There are only two services in the accessibility API, which have to be supported by any accessible object.

The com.sun.star.accessibility.Accessible service contains the com.sun.star.accessibility.XAccessible interface and must be supported by every UNO object that is accessible.

The com.sun.star.accessibility.AccessibleContext service contains the com.sun.star.accessibility.XAccessibleContext interface and, optionally, the com.sun.star.accessibility.XAccessibleEventBroadcaster interface. This service must be supported by every accessible object that is returned by the com.sun.star.accessibility.XAccessible:getAccessibleContext() function.

17.7 Using the Accessibility API

When you are writing your own ATs and want to use the UNO accessibility API directly, you must first connect to OpenOffice.org. Connecting to OpenOffice.org is explained elsewhere in this document. Once a connection is established, the toolkit with its com.sun.star.awt.XExtendedToolkit interface can be used to retrieve a list of all currently open top-level windows. From these, you can then get the accessible root nodes of the accessibility object trees associated with the windows. When you register an com.sun.star.awt.XTopWindowListener you will then be informed about new top-level windows, as well as top-level windows that have disappeared.

With the top-level accessible objects at hand, you can use the Java version of the accessibility API as it is described in detail in the following sections. To be informed about focus changes—so that, for example, a screen reader can track the currently focused object and read it to the user—an AT has to register all non-transient objects of an accessibility tree.

The general operation of a simple AT consists of the following steps:

1. Connect to OpenOffice.org.
2. Retrieve the currently visible top-level windows and register as top window listener to keep the list up-to-date.
3. Traverse the trees by getting their root elements from each window. See the description of the com.sun.star.accessibility.XAccessibleContext for a code example for this.
4. Register each accessible object as com.sun.star.accessibility:XAccessibleEventListener.
5. If called back with an com.sun.star.accessibility:AccessibleEventObject object, then process two kinds of events:
   - Events that denote a state change with either OldValue or NewValue containing the FOCUSED constant indicate that the source object of this event got either focused or unfocused.
   - When receiving events of type CHILD, register as listener at the object that is specified by the event, as well as all of the object's children.

17.7.1 A Simple Screen Reader

To illustrate the use of the UNO accessibility API, we will describe how to implement a simple AT. The simple screen reader, or SSR, will display some information about the currently focused object. As you can see in Illustration 209, the SSR consists of three windows. The bottom window logs all events that the SSR receives from any of the accessibility objects to which it is registered.

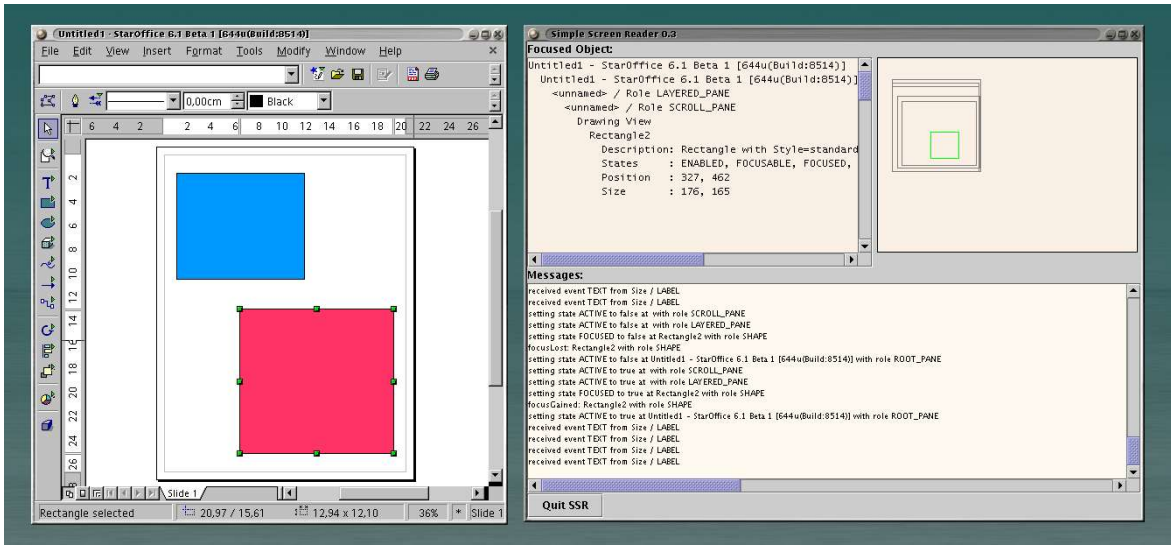


Illustration 209: The simple screen reader shows information on the currently focused object in two different views. A textual display on the top left shows its description, states and bounding box as well as the names of its ancestors. The graphical view on the top right shows the bounding boxes of the object itself (green) and its ancestors (gray). The bottom window logs all events sent by any of the accessibility objects.

The two upper windows display information about the currently focused object, which in this screen shot is a shape in a presentation document. The left window displays the names of all the ancestors of the focused object and the path from that object to the root object of the accessibility tree. The left window also displays the focused object's description, its state, its location, and its size on the screen. In this example, the focused object is named “Rectangle2”, which corresponds to the red rectangular shape. Its parent is called “Drawing View” and the root of the tree has the name of the document, which is “Untitled1”, followed by the product name and some debug information.

The upper right window displays similar information graphically. The focused object is shown as a green rectangle, while its ancestors are drawn as gray rectangles. You can see how the objects are nested. This corresponds to the requirement that the bounding of child objects must not overlap that of their parents. Note that the blue rectangular shape is not visible in this window, because it is a sibling of the focused red rectangle, but does not lie on that object's path to the root of the root object. Also note that some of the rectangles are off-center and smaller than they should be. This is because the rectangles that represent accessible objects, which in turn represent part of the GUI, are drawn with their screen location and size relative to the whole screen; the outermost rectangle that is enclosed by the gray background represents the screen of which the screen shot shows only a part.

The bottom window logs all the events that the SSR receives from the accessible objects it has register as event listener at.

Features

The SSR was designed to be a simple program that illustrates how to use the UNO accessibility API. However, we have not always chosen the most simple way to do something. There are therefore some features that may be useful in the “real” AT:

- The SSR has a background connection timer task that waits for a OpenOffice.org application to start and then connects automatically to that application.

- It uses independent threads to register as listener at a whole accessibility (sub-) tree of new top-level application windows or newly created accessible objects. The same is true for removing the listener from windows not visible anymore or accessible objects that are removed from their tree.
- There are two different views that display information about the currently focused object. This illustrates how information of a certain accessibility object is retrieved by using different interfaces of the accessibility API.
- A message area shows all received events regardless of whether they are necessary to keep track of the currently focused object. With this, the SSR serves as simple event monitor as well.

Class Overview

The SSR is implemented with the following classes and interfaces:

SSR
This is the main class of the tool. It is responsible for setting up and managing the GUI

ConnectionTask
The connection task is a thread that waits in the background for a OpenOffice.org application to start. As soon as there is one, it connects to it and initiates the registration at all of its accessible objects.

RegistrationThread
Each object of this class creates one thread that adds or removes the event listener at all accessible objects in one tree, with one tree per application window. This is done in separate threads so that the normal operation of the tool is not blocked while registering to complex applications with many objects.

EventListenerProxy
This is a singleton class. Its only instance is registered as listener at all accessible objects. It runs in its own thread and delivers the received events eventually to the actual event handler. This de-coupling between event reception and event handling is necessary to avoid deadlocks. Soon this will become obsolete.

EventHandler
There is usually only one object of this class. It prints log messages for all the events it receives and provides special handling some of events:

- Top window events denoting new or removed application windows are not accessibility events in a strict sense. They have to be listened to, however, to add or remove the event listener to the accessibility objects that correspond to these windows.
- State events that inform the listener of a set or reset `FOCUSED` state. This is the most important event for the SSR in order to keep track of the currently focused object.
- Events that signal a change of the geometric property of the currently focused object trigger a redisplay of the two windows that display that object. This ensures that you always see the current position of the object.

TextualDisplay
This widget displays textual information about the focused object as described previously.

GraphicalDisplay
This widget displays graphical information about the focused object as described previously.

IaccessibleObjectDisplay
: This is the interface supported by the two display widgets. It defines how the event handler tells the displays about focus and geometry changes. You can add other displays as well by adding a widget that implements this interface to the event handler and to the GUI.

MessageArea
: The message area at the bottom is a simple text widget that scrolls its content so that the last line that contains the most recent message is always visible.

NameProvider
: This is a useful helper class that converts numerical IDs into names, roles, events, or states.

Parts of some of these classes will be explained at later points in this text. Others, like the `ConnectionTask` class, are a mere technicality with respect to the accessibility API and will not be detailed any further.

Putting the Accessibility Interfaces to Work

Once an accessible object has been obtained by a call to `com.sun.star.accessibility.XAccessible:getAccessibleContext()`, you can switch between the interfaces belonging to the accessibility API by using the usual UNO cast mechanisms. There is, however, no way back to the object from which the accessible object has been obtained through procedures provided by the accessibility API.

XAccessibleContext

`com.sun.star.accessibility.XAccessibleContext` is the central interface of the accessibility API. In addition to the hierarchy information described previously, it provides access to some important information. The functions `getAccessibleRole()`, `getAccessibleName()` and `getAccessibleDescription()` return descriptions of the object in increasing detail:

Role
: The role classifies all accessibility objects into a handful of different classes. Most roles are taken from the Java accessibility API, such as `PUSH_BUTTON`, `RADIO_BUTTON`, `SCROLL_BAR` or `TEXT`. Some have been defined for the accessibility API so that, in addition to GUI elements, documents can be made accessible. These roles are `DOCUMENT` for document windows or views, `PARAGRAPHS` for text sections, or `SHAPE` for graphical objects. Roles are described in the `com.sun.star.accessibility.AccessibleRole` constants group.

Name
: Names allow you to distinguish between objects with the same role. For example, the buttons at the bottom of a dialog all have the role `PUSH_BUTTON`. Their names may be “OK”, “Cancel” or “Help”. Where necessary, names are made unique with respect to the object's siblings. Names for shapes can be “Rectangle 0”, “Ellipse 1”, “Rectangle 2”, or “Curve 3”.

Description
: To further describe the purpose of accessibility objects, description strings are provided. Descriptions of shapes can contain their style and some properties whose values differ from that style. If you have changed the color of a rectangle to red, its description may look like “Rectangle with style=default and color=red”.

Names and descriptions are strings localized according to the locale returned by `getAccessibleLocale()`.

Two functions of the interface have not been mentioned so far. The function `getAccessibleRelationSet()` returns the set of relations defined for an accessibility object. Likewise `getAccessibleStateSet()` returns a set of states that are active for an object.

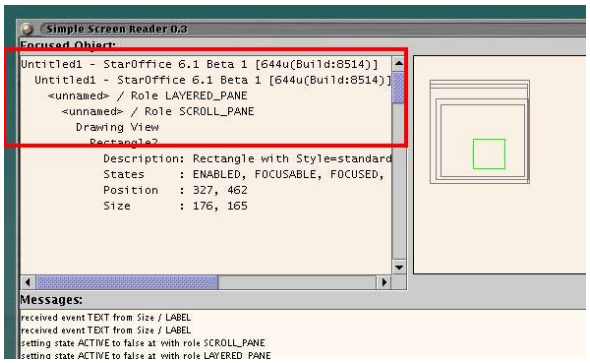


Illustration 210: Path in the SSR

The `showParents()` method from the `TextualDisplay` class of the SSR displays the path from a given object to the root of the accessibility tree by printing each object's accessible name indented relative to its father.

```
private String showParents (XAccessibleContext xContext) {
```

The method first obtains references to all the objects that belong to this path.

```
    Vector aPathToRoot = new Vector();
    while (xContext != null) {
        aPathToRoot.add (xContext);
        // Go up the hierarchy one level to the object's parent.
        try {
            XAccessible xParent = xContext.getAccessibleParent();
            if (xParent != null)
                xContext = xParent.getAccessibleContext();
            else
                xContext = null;
        }
        catch (Exception e) {
            System.err.println ("caught exception " + e + " while getting path to root");
        }
    }
```

This is done in two steps. First, a call to the `getAccessibleParent()` method to get the parent of the object and thereby moving to the previous level in the accessibility tree. Second, from the returned `XAccessible` reference, the accessible context is retrieved by calling `getAccessibleContext()`. This is repeated until an object is reached that has no parent and `getAccessibleParent()` returns `null`.

The path of the accessibility tree is now printed by appending text to the `msTextContext` member variable, which later is displayed in a `JtextArea` widget. To cope with accessibility objects that return an empty name, the role of these objects is used to represent them. Note how the indentation string is updated after every object by appending the `msIndentation` member.

```
    String sIndentation = new String ();
    for (int i=aPathToRoot.size()-1; i>=0; i--) {
        XAccessibleContext xParentContext = (XAccessibleContext)aPathToRoot.get(i);
        String sParentName = xParentContext.getAccessibleName();
        if (sParentName.length() == 0)
            sParentName = "<unnamed> / Role "
                + NameProvider.getRoleName(xParentContext.getAccessibleRole());
        msTextContent += sIndentation + sParentName + "\n";
        sIndentation += msIndentation;
    }
```

The indentation is returned so that further output can be properly aligned.

```
    return sIndentation;
}
```

XAccessibleComponent

The `XAccessibleComponent` interface gives access to geometric properties, such as size and position on the screen. This interface should be implemented by every object that has a visible representation, that is, by all objects that are not simple containers.

The coordinates used by the functions of this interface are returned and and are expected in pixel values and not, as is elsewhere in the UDK, in internal coordinates (100^{th} of mm). There are three different origins to which coordinates may be specified:

- Relative to an object's bounding box. This is relative to the object itself. Used by the `contains()` and `getAccessibleAt()` functions.
- Relative to an object's parent. Used by the `getBounds()` and `getLocation()` functions.
- Absolute or relative to the screen origin. Used by the `getLocationOnScreen()`.

Because all three coordinate systems are based on pixel values, the `getSize()` function is independent of the coordinate system.

The bounding rectangle that encloses the visual presentation of an object can be retrieved by calling `getBounds()`. If you only need the location or the size, then call `getLocation()` or `getSize()`. The function `getLocationOnScreen()` returns the absolute screen coordinates.

There or two functions that determine whether the bounding boxes of the object or one of its children contain a given test point. The function `contains()` checks whether the test point lies within the bounding box of the object. Children can be tested with `getAccessibleAt()`. When one of the direct children contains the test point, a reference to this object is returned.

In addition to the geometrical functions, there are the two `getForeground()` and `getBackground()` functions that describe an object's appearance. Keep in mind that an object does not necessarily have a monochrome background color. There can be a hatching, gradient, or bitmap as well. The returned background color in this case is an approximation.

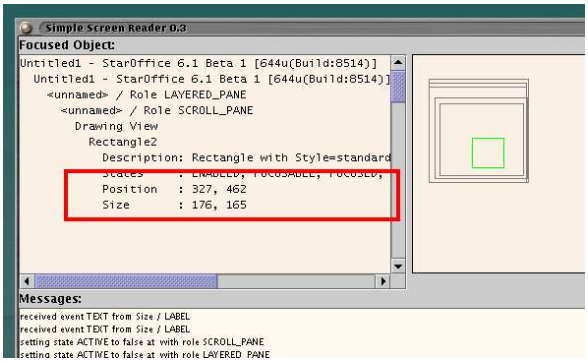


Illustration 211: Geometrical information in the SSR

The `showComponentInfo()` method of the `TextualDisplay` class takes as argument a reference to the accessible object for which geometrical information is shown, as well as the indentation string computed in the `showParents()` method.

```
private void showComponentInfo (XAccessibleContext xContext, String sIndentation) {
```

When given an `XAccessibleContext` reference, you must cast it to a `XAccessibleComponent` reference in order to access geometrical information about an accessible object.

```
    XAccessibleComponent xComponent = (XAccessibleComponent)UnoRuntime.queryInterface(
            XAccessibleComponent.class, xContext);
    if (xComponent != null) {
```

If the cast was successful, then simply call the `getLocationOnScreen()` and `getSize()` methods to obtain the object's bounding box in screen coordinates. If `getBounds()` is called instead, the parent's screen coordinates must be added to obtain an absolute position.

```
        Point aLocation = xComponent.getLocationOnScreen();
        msTextContent += sIndentation + "Position   : "
            + aLocation.X + ", " + aLocation.Y + "\n";

        Size aSize = xComponent.getSize();
        msTextContent += sIndentation + "Size       : "
            + aSize.Width + ", " + aSize.Height + "\n";
    }
}
```

XAccessibleExtendedComponent

While the `com.sun.star.accessibility.XAccessibleComponent` interface should be implemented by almost every accessible object, the support of the `XAccessibleExtendedComponent` interface is optional. Its most important function, `getFont()` returns the font used to display text.

The `getTitledBorderText()` function returns the text that is displayed on an object's window borders, which in the case of OpenOffice.org is only relevant for top-level windows.

The `getToolTipText()` function returns the tool tip text that is displayed when the mouse pointer rests long enough over one point of the object.

XAccessibleText

The interface `XAccessibleText` handles read-only text. It serves three purposes: to obtain certain parts of a text, to obtain the text's size and location on the screen, and to handle the caret position. An accessibility object that implements this interface usually represents only a part of a larger text. Typically, this is a single or a small number of paragraphs. You can use the relation types `com.sun.star.accessibility.AccessibleRelationType:CONTENT_FLOWS_FROM` and `com.sun.star.accessibility.AccessibleRelationType:CONTENT_FLOWS_TO` to explicitly represent the text flow from one text part to another. Without these relations, the text flow has to be determined from the structure of the accessibility tree alone.

Selection

Represented text may contain a selected text portion, which is typically displayed highlighted (inverse). There are four functions to access and modify the selection. The selected text, as well as its start and end index, can be accessed with the `com.sun.star.accessibility.XAccessibleText:getSelectedText()`, `com.sun.star.accessibility.XAccessibleText:getSelectionStart()` and `com.sun.star.accessibility.XAccessibleText:getSelectionEnd()` functions respectively. To modify the selection, call `com.sun.star.accessibility.XAccessibleText:setSelection()` with the new start and end indices.

Text type

The functions `com.sun.star.accessibility.XAccessibleText:getTextAtIndex()`, `com.sun.star.accessibility.XAccessibleText:getTextBeforeIndex()` and `com.sun.star.accessibility.XAccessibleText:getTextBehindIndex()` return parts of the text where the part or length of text returned is specified by the *text type*. Defined in the `com.sun.star.accessibility.AccessibleTextType` constants collection, some of these text types further explanation:

LINE

The type `LINE` indicates all text on a single line as it is displayed on the screen. This may include a hyphen at the line end. The hyphen is not actually part of the text, but is visible on the screen. They are only included in text parts of type `LINE`.

CHARACTER, GLYPH

Glyph is used in this context to mean that everything between two adjacent cursor positions is considered to be a glyph. In Thai, for example, you can stack up to four (Unicode) characters upon each other. When moving with the cursor keys, those character groups are interpreted as single glyphs, and the cursor can only be set in front or after such glyphs, but not inside.

WORD, SENTENCE, PARAGRAPH

The definition of words, sentences and paragraphs are implementation and locale dependent.

ATTRIBUTE_RUN

An attribute run is a sequence of characters of maximal length, where all characters have the same attributes. For example the text "*This* ***is an*** **example**" consists of three attribute runs: "*This* ", "***is an***" and " **example**". The first and last one each have only a single attribute, italic respectively bold. The second run has both attributes at the same time. Note that all three spaces belong to one of the attribute runs.

There are other functions to access text. To retrieve the whole text represented by an object, use `getText()`. Call `getTextRange()` to get only a part of text between and including two given indices. A single character can be accessed with the `getCharacter()` function.

To copy a text range to the clipboard, call the `copyText()` function. See also the related `cutText()` and `pasteText()` functions of the `com.sun.star.accessibility.XAccessibleEditableText` interface.

Caret and Text Indices

The caret is often referred to as the cursor and is typically displayed as a vertical line. On the screen, it is always placed between two adjacent glyphs, or at the beginning or the end of a text line. When specifying its position in terms of character positions, the position of the character to its right is used. Thus, the index 0 says that the caret is at the very beginning of the text. When the caret is located behind the last character, then its position equals the text length, that is, the number of characters of the text as returned by `getCharacterCount()`.

When the caret changes its position, an event must be sent to all listeners so that the old and new position can be indicated.

The caret position is returned by the `getCaretPosition()` function, and can be set by calling `setCaretPosition()`.

Other index-related functions are `getCharacterAttributes()`, which returns the attributes at the specified index, `getCharacterBounds()`, which returns the bounding box of the character at the specified index, and `getIndexAtPoint()`, which returns the index of the character at the specified position.

XAccessibleEditableText

The `XAccessibleEditableText` interface extends the `XAccessibleText` interface by adding functions that let you modify the text. The interface is therefore only implemented when the text represented by the implementing object is readable *and* writeable.

With the functions `deleteText()`, `insertText()` and `replaceText()`, you can delete, insert, and replace text. The `setText()` function is a special case of `replaceText()` and replaces the whole text at once.

The `cutText()` and `pasteText()` functions, together with `copyText()`, from the `com.sun.star.accessibility.XAccessibleText` interface provide access to the clipboard.

Finally, the `setAttributes()` function is the counterpart of `getCharacterAttributes()` in the `com.sun.star.accessibility.XAccessibleText` interface. With this function, you can replace the existing attributes with the given set. To add one attribute, first use `getCharacterAttributes()` to get the current set of attributes, add the attribute to that set, and finally call `setAttributes()` to set the new set of attributes.

XAccessibleTable

The `com.sun.star.accessibility.XAccessibleTable` interface represents two-dimensional tables of data. The Calc application is one example of its implementation. It grants access to individual cells or groups of cells.

Global information about a table can be accessed with two functions: `getAccessibleCaption()` returns the caption and `getAccessibleSummary()` returns a summary describing the content of a table.

A table is organized in horizontal rows and vertical columns. The number of rows and columns—and indirectly the number of cells—can be determined by calling the functions `getAccessibleRowCount()` and `getAccessibleColumnCount()`. Here, in contrast to the general rule of only giving access to visible objects, all cells are represented by a table. This exception is necessary to stay consistent with Java tables.

information on rows and columns is returned by the `getAccessibleRowDescription()` and `getAccessibleColumnDescription()` functions. Note that both functions return objects implementing the `com.sun.star.accessibility.XAccessibleTable` interface themselves. The headers of rows and columns can be retrieved by calling `getAccessibleRowHeaders()` and `getAccessibleColumnHeaders()`.

To obtain a reference to a certain cell specified by its row and column indices, use `getAccessibleCellAt()`. A table cell may span multiple rows and columns. You can determine the number of rows and columns that a cell spans with the `getAccessibleRowExtentAt()` and `getAccessibleColumnExtentAt()` functions.

Selections in tables can have two different forms. You can have a multi-selection of rectangular areas of single or multiple cells, or you can select whole rows and columns. The function `isAccessibleSelected()` determines whether a single cell that spans a position specified by a row and a column index is selected. To determine whether certain rows or columns are selected, use the `isAccessibleRowSelected()` and `isAcessibleColumnSelected()` functions. Finally, the functions `getSelectedAccessibleRows()` and `getSelectedAccessibleColumns()` each return a sequence of indices of the currently selected row and columns.

There are three functions that can be used to switch between cell indices and row and column indices. Cell indices are the same as the child indices used by the `getAccessibleChildCount()` and `getAccessibleChild()` functions of the `com.sun.star.accessibility.XAccessibleContext` interface. Row and column indices have been used previously, and specify each cell by stating its row and column. The `getAccessibleIndex()` function returns the corresponding cell index for a given row and column index. The `getAccessibleRow()` and `getAccessibleColumn()` functions return the corresponding row and column index, respectively, for a given cell index.

XAccessibleEventBroadcaster

Most accessible objects need to broadcast events that describe changes of its internal state or visual appearance—this can be done with the `com.sun.star.accessibility.XAccessibleEventBroadcaster`interface. If you want to be informed of such changes, you can register a listener with the `addEventListener()` function, or remove it with `removeEventListener()`.



Be sure to cast an object reference to the `XAccessibleEventBroadcaster` interface before calling these functions, because there are other broadcaster interfaces with functions of the same name.

The SSR registers the event listener in separate threads. The major work is done by a method called `traverseTree()` that takes an accessible context and traverses the whole tree rooted in this object.

```
public long traverseTree (XAccessibleContext xRoot) {
```

After casting the context reference to a reference of the `XAccessibleEventBroadcaster` interface, it either adds or removes the listener `maListener` at the accessibility object.

```
    long nNodeCount = 0;
    if (xRoot != null) {
        // Register the root node.
        XAccessibleEventBroadcaster xBroadcaster =
            (XAccessibleEventBroadcaster) UnoRuntime.queryInterface (
                XAccessibleEventBroadcaster.class,
                xRoot);
        if (xBroadcaster != null) {
            if (mbRegister)
                xBroadcaster.addEventListener (maListener);
            else
                xBroadcaster.removeEventListener (maListener);
            nNodeCount += 1;
        }
```

Once the given object is handled, the traversing of the tree continues by calling this method recursively for every child.

```
        try {
            int nChildCount = xRoot.getAccessibleChildCount();
            for (int i=0; i<nChildCount; i++) {
                XAccessible xChild = xRoot.getAccessibleChild (i);
                if (xChild != null)
                    nNodeCount += traverseTree (xChild.getAccessibleContext());
            }
        }
```

Because the iteration over the direct children of the given accessible context may take a while, there is the possibility that some of the children do not exist anymore. It is therefore important to catch `IndexOutOfBoundsException` and `DisposedException` exceptions. Note that this is not a perfect solution, because children that are added to the given object are not handled properly. A better algorithm would listen to events of the new and removed children of the object to which it was recently registered:

```
        catch (com.sun.star.lang.IndexOutOfBoundsException aException) {
            // The set of children has changed since our last call to
            // getAccesibleChildCount().  Don't try any further on this
            // sub-tree.
        }
        catch (com.sun.star.lang.DisposedException aException) {
            // The child has been destroyed since our last call to
            // getAccesibleChildCount().  That is OK. Don't try any
            // further on this sub-tree.
        }
    }
    return nNodeCount;
}
```

This method keeps track of how many objects it has added to the listener. This number is used in the `run()` method to write an informative message.

```
public void run () {
    System.out.println ("starting registration");
```

```
    long nNodeCount = traverseTree (mxRoot);
    System.out.println ("ending registration");
    if (mbShowMessages) {
        if (!mbRegister)
            MessageArea.print ("un");
        MessageArea.println ("registered at " + nNodeCount
            + " objects in accessibility tree of " + mxRoot.getAccessibleName());
    }
}
```

XAccessibleEventListener

The interface `com.sun.star.accessibility.XAccessibleEventListener` is the counterpart to the accessible event broadcaster, and is called for every change of any accessible object at which it has been registered. The `notifyEvent()` function is called with an `com.sun.star.accessibility.AccessibleEventObject` structure. That structure comprises four fields: the `EventId` field, which is one of the `com.sun.star.accessibility.AccessibleEventId` constants, describes the type of the event. The object that sent the event is referenced from the Source field. The `OldValue` and `NewValue` fields contain event type specific values that contain the changed value before and after the modification took place. The type of content that is expected in the `OldValue` and `NewValue` fields is explained together with the `AccessibleEventId` event types.

The `notifyEvent()` method of the `EventHandler` class is not called directly from the accessibility objects. There is an instance of the `EventListenerProxy` class in between. That class simply forwards the events at the right time and is therefore not explained in more detail here.

```
public void notifyEvent (com.sun.star.accessibility.AccessibleEventObject aEvent) {
```

The one event type that is covered here is the `CHILD` event, which is sent when a new accessibility object has been created, or an existing one has been removed.

```
    // Guard against disposed objects.
    try {
        switch (aEvent.EventId) {
            case AccessibleEventId.CHILD:
                handleChildEvent (
                    objectToContext (aEvent.OldValue),
                    objectToContext (aEvent.NewValue));
                break;
```

The handling of the rest of the event types is omitted here to keep this explanation simple.

Again, it is important to guard against the possibility of events arriving after the object they were sent for has been destroyed. For this simple tool, it is sufficient to silently ignore the resulting exception.

```
        }
    }
    catch (com.sun.star.lang.DisposedException e) {
    }
}
```

Before showing you the actual handling of child events, you can look at the `objectToContext()` method that is used to convert the `OldValue` and `NewValue` fields of the event structure from UNO Anys to `XAccessibleContext` references. It takes a weakness of the accessibility API IDL specification into account: the type of the content of the `Source` field is not explicitly stated. As a result, it contains references to both the `XAccessible` and the `XAccessibleContext` interfaces. To cope with this, the conversion method first uses the `com.sun.star.uno.AnyConverter` class to retrieve an `XAccessible` reference from the event source.

```
private XAccessibleContext objectToContext (Object aObject) {
    XAccessibleContext xContext = null;
    XAccessible xAccessible = null;
    try {
        xAccessible = (XAccessible)AnyConverter.toObject(
            new Type(XAccessible.class), aObject);
    }
    catch (com.sun.star.lang.IllegalArgumentException e) {
    }
```

If that was successful, the accessible context from the object is returned.

```
    if (xAccessible != null)
        xContext = xAccessible.getAccessibleContext();
```

If retrieving an `XAccessible` reference from the event's source field failed, this is repeated directly with the `XAccessibleContext` interface.

```
    else
        try {
            xContext = (XAccessibleContext)AnyConverter.toObject(
                new Type(XAccessibleContext.class), aObject);
        }
        catch (com.sun.star.lang.IllegalArgumentException e) {
        }
    return xContext;
}
```

Handling the child event itself is comparably simple: create a new `RegistrationThread` object that adds (or removes) the listener to the object and all of its children.

```
private void handleChildEvent (XAccessibleContext aOldChild, XAccessibleContext aNewChild) {
    if (aOldChild != null)
        // Remove event listener from the child and all of its descendants.
        new RegistrationThread (maListenerProxy, aOldChild, false, false);
    else if (aNewChild != null)
        // Add event listener to the new child and all of its descendants.
        new RegistrationThread (maListenerProxy, aNewChild, true, false);
}
```

XAccessibleSelection

While the `com.sun.star.accessibility.XAccessibleText` and `com.sun.star.accessibility.XAccessibleTable` interfaces already support selection of text and table cells, respectively, there is a special interface for the general case. The `XAccessibleSelection` interface manages a sub-set of an object's children that form the selection. The number of selected children is returned by `getSelectedAccessibleChildCount()`, which, of course, is smaller than or equal to the total number of children as returned by `getAccessibleChildCount()` of the `com.sun.star.accessibility.XAccessibleContext` interface. The selected children can be retrieved by calling the `getSelectedAccessibleChild()` function. Note that the same index passed to `getSelectedAccessibleChild()` and to `com.sun.star.accessibility.XAccessibleContext:getAccessibleChild()` will generally return different objects.

The selection can be modified with various functions. The functions `selectAllAccessible()` and `clearAccessibleSelection()` select or deselect, respectively, all of the children. To select or deselect a single child, use `selectAccessibleChild()` or `deselectSelectedAccessibleChild()`. Whether a child belongs to the selection can be determined by calling the `isAccessibleChildSelected()` function.

Each child that belongs to the selection is expected to have the `com.sun.star.accessibility.AccessibleStateType:SELECTED` state set. When the selection changes, two kinds of events are expected to be broadcast. One for each child that is selected or deselected that tells the listeners about the toggled `SELECTED` state, and one event from their parent that informs the listeners of the modified selection as represented by the `com.sun.star.accessibility.XAccessibleSelection` interface.

XAccessibleRelationSet

In addition to the parent-child relationship that defines the accessibility object tree, each accessible object may have one or more relations to other accessible objects, independent of that hierarchy. The types of possible relations are defined and explained in the `com.sun.star.accessibility.AccessibleRelationType` set of constants.

One example is the com.sun.star.accessibility.AccessibleRelationType:LABEL_FOR and com.sun.star.accessibility.AccessibleRelationType:LABELED_BY pair of relations that is used to express the case where one object is the label for a one or more controls of the GUI. The label and its controls may be spatially adjacent to make their relationship clear to the sighted user. But in the accessibility object tree, they may belong to different sub-trees. With a set of relations, these objects can be linked together.

Each relation is an com.sun.star.accessibility.AccessibleRelation structure that contains the relation type and a set of references to its target objects. In the previous example, there would be one com.sun.star.accessibility.AccessibleRelationType:LABEL_FOR relation for the label with all its controls in the target set and one com.sun.star.accessibility.AccessibleRelationType:LABELED_BY relation from each control to the label.

For each relation type there may be one relation belonging to a relation set as represented by the com.sun.star.accessibility.XAccessibleRelationSet interface. Therefore, the number of relations that is returned by getRelationCount() can not be greater than the number of relation types. Relations can be accessed either by calling the com.sun.star.accessibility.AccessibleRelation:getRelation() function with an index, or by calling the com.sun.star.accessibility.AccessibleRelation:RelationType() function with a relation type. Note that there are no fixed mappings from relation types to indices. You can test whether a relation set contains a relation of a given type by using the com.sun.star.accessibility.AccessibleRelation:containsRelation() function.



The relation set returned by com.sun.star.accessibility.XAccessibleContext:getAccessibleRelationSet() returns a copy of the relation set of an accessible object. Modifying that copy does not change the relation set of the object.

XAccessibleStateSet

An accessible object can be in one or more states, which are available through the interface com.sun.star.accessibility.XAccessibleStateSet. An object that is currently focused will have the FOCUSED state set. One that belongs to the selection of its parent has the SELECTED state set. Independent of the current focus and selection, such an object would also have the states FOCUSABLE and SELECTABLE set to indicate that it *may* be focused and *may* be selected. All states are described in the com.sun.star.accessibility.AccessibleStateType constants collection.

Call the com.sun.star.accessibility.XAccessibleStateSet:isEmpty() function to query whether a state set has no state set at all. To query a state set for one or more states, use the com.sun.star.accessibility.XAccessibleStateSet:contains() and com.sun.star.accessibility.XAccessibleStateSet:containsAll() functions respectively. A sequence containing all the set states is returned by the com.sun.star.accessibility.XAccessibleStateSet:getStates() function.



The state set returned by com.sun.star.accessibility.XAccessibleContext:getAccessibleStateSet() returns a copy of the state set of an accessible object. Modifying that copy does not change the states of the object. Therefore, modifying a state set is not useful and consequently no modifying functions are supported by the com.sun.star.accessibility.XAccessibleStateSet interface. States are set or reset indirectly by using the standard UNO interfaces or the GUI.

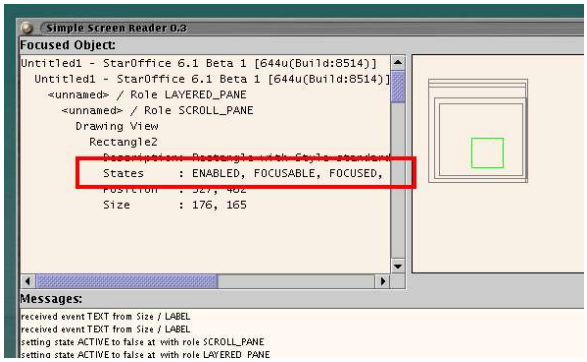


Illustration 212: List of states in the SSR

The `showStates()` method of the `TextualDisplay` class of the SSR first obtains a state set object from the given accessible context and prints a list of all the states contained therein by using the `getStates()` method to convert the state set into an array of state IDs. It then iterates over all IDs and uses the `NameProvider` class to convert the numerical IDs into human readable strings.

```
private void showStates (XAccessibleContext xContext, String sIndentation) {
    // Get the state set object...
    XAccessibleStateSet xStateSet = xContext.getAccessibleStateSet();
    // ...and retrieve an array of numerical IDs.
    short aStates[] = xStateSet.getStates();

    // Iterate over the array and print the names of the states.
    msTextContent += sIndentation + "States     : ";
    for (int i=0; i<aStates.length; i++) {
        if (i > 0)
            msTextContent += ", ";
        msTextContent += NameProvider.getStateName(aStates[i]);
    }
    msTextContent += "\n";
}
```

XAccessibleValue

In a typical GUI, there are many controls whose characteristic feature can be represented by a single numerical value. Examples are spin boxes, scales and sliders, or combo boxes that contain only numbers, such as the font size. These objects should support the `XAccessibleValue` interface.

The current value can be read and set with the `getCurrentValue()` and `setCurrentValue()` functions. The valid range of values is returned by the `getMaximumValue()` and `getMinimumValue()` functions.

XAccessibleImage

The main purpose of the `XAccessibleImage` interface is to be an indicator that an object represents an image or a bitmap. The functions of this interface do not add functionality that is not already present in the `com.sun.star.accessibility.XAccessibleContext` interface.

The `getAccessibleImageDescription()` function returns a localized description of the image. The functions `getAccessibleImageHeight()` and `getAccessibleImageWidth()` return the size of the image in pixels.

XAccessibleAction

With the `XAccessibleAction` interface, an accessible object can provide access to actions that can be performed on the object. These actions may or may not correspond to actions that are already available over the GUI.

The number of available actions is returned by `getAccessibleActionCount()`. To execute an action call `doAccessibleAction()` with the index of the desired action. The description of an action is returned by the `getAccessibleActionDescription()`.

The `getAccessibleKeyBinding()` function tells you what key bindings exist for a certain action. See the description of the `com.sun.star.accessibility.XAccessibleKeyBinding` interface below.

XAccessibleKeyBinding

The purpose of the `XAccessibleKeyBinding` interface is to represent an arbitrary set of key strokes. When a key binding is associated with an action (see the description of the interface above) then each of its key strokes executes that action. A key stroke itself consists of one or more keys to be pressed. See `com.sun.star.awt.KeyStroke` for details.

XAccessibleHypertext

The interface `com.sun.star.accessibility.XAccessibleHypertext` is derived from `com.sun.star.accessibility.XAccessibleText`. This interface represents text that contains hyperlinks. Those hyperlinks are represented by `com.sun.star.accessibility.XAccessibleHyperlink` objects, which are described next.

To iterate over all hyperlinks in a text, use `getLinkCount()` to determine the number of links. The `getLink()` function returns the `com.sun.star.accessibility.XAccessibleHyperlink` object for a specific index. If you want to know whether there is a link at a certain text position, use the `getLinkIndex()` function to obtain the corresponding object. When the returned reference is empty then there is no hyperlink at that position.

XAccessibleHyperlink

Hyperlinks contained in a hypertext document are modeled by the `XAccessibleHyperlink` interface. In its simplest form, a hyperlink corresponds to an HTML link. However, there may be more complex hyperlinks where there is more than one action assigned to a single hyperlink. An example of this is HTML image maps. To give access to the actions, the interface is derived from the `com.sun.star.accessibility.XAccessibleAction` interface. In addition to the information provided by the `com.sun.star.accessibility.XAccessibleAction` interface, you can request an action's anchor and object. A typical return value of the `getAccessibleActionAnchor()` function for HTML links would be the text between the `<a href...>` and `</a>` tags. Likewise, the `com.sun.star.accessibility.XAccessibleHyperlink:getAccessibleActionObject()` function returns a HTML link's URL.

A hyperlink specifies its text relative to the enclosing hypertext by providing a start- and end index through the `getStartIndex()` and `getEndIndex()` functions. You can ask a link about the validity of the referenced target by calling its `isValid()` function. Note that this state is volatile and may change without notice.

Appendix A: OpenOffice.org API-Design-Guidelines

The following rules apply to all external programming interface specifications for OpenOffice. The API consists of the following stereotypes or design elements:

Structures
Structures are used to specify simple composed data elements.
(Structures only consist of data, not methods.)

Exceptions
Exceptions are used for error handling.
(Exceptions can be thrown or transported using an `any`.)

Interfaces
Interfaces are used to specify a single aspect in behavior.
(Interfaces only consist of methods, not data.)

Services
Services are used to specify abstract objects.
(Services specify properties and the interaction of the supported interfaces.)

Typedefs
Typedefs are used to define basic types for specific purposes.
(This stereotype should be used carefully.)

A.1 General Design Rules

These rules describe basic concepts used in OpenOffice.org API design. They are mandatory for all OpenOffice.org contributions. They are recommended good practice for development of third-party software.

A.1.1 Universality

It is preferable to design and use universal interfaces instead of specialized ones. Interface reuse should prevail. Whenever a new interface is about to be created, consider the possibility of similar requirements in other application areas and design the interface in a general manner.

A.1.2 Orthogonality

The functionality of interfaces should extend each other. Avoid redundancy , but if it leads to a major simplification for application programmers, proceed. In general, functionality that can be acquired from basic interfaces should not be added directly. If necessary, create an extra service which provides the functionality and works on external data.

A.1.3 Inheritance

All interfaces are derived from `com.sun.star.uno.XInterface`. Other superclasses are only allowed if the following terms are true:

- the derived interface is a direct extension of the superclass
- the superclass is necessary in every case for the interface and inheritance if it is logical for the application programmer
- the superclass is the only possible superclass due to this definition

A.1.4 Uniformity

All identifiers have to follow uniform rules for semantics, lexical names, and order of arguments. Programmers and developers who are familiar with any portion of the API can work with any other part intuitively.

A.1.5 Correct English

Whoever designs API elements is responsible for the correct spelling and meaning of the applied English terms, especially for lexical names of interfaces, methods, structures and exceptions, as well as members of structures and exceptions. If not absolutely certain, use Merriam-Webster's Dictionary (http:// www.m-w.com). We use U.S. spelling.

Mixed capitalization or underscores (the latter only for lexical constants and enumeration values) are used to separate words within single identifiers. Apply the word separation appropriately. English generally does not have compound words, unlike, for example, German.

A.2 Definition of API Elements

In this chapter, several API elements are defined, and how they are used and the rules that apply.

A.2.1 Attributes

Attributes are used to access data members of objects through an interface.

Naming

Attributes are defined in interfaces as get and optional set methods. Although UNOIDL knows attributes for compatibility reasons, this feature is not used. The attribute identifier begins with a capital letter. The mixed upper and lower case method is used to separate single words. Only letters and numbers are allowed with no underscores, for example, `getParent()` and `setParent()`.

Usage

Attributes are used to express structural relationships, with and without lifetime coupling. For scripting languages, the attributes are accessed as properties.

A.2.2 Methods

Methods are functions defined within an interface. Technically, an interface only consists of methods. There is a syntax for attributes, but these map to methods.

Naming

Method identifiers begin with a verb in lowercase, for example, close, and continue with initial caps, that is, the first letter of each word is capitalized with no underscores. For example, `getFormat()`.

Method names consisting of a verb without any additional terms can only be used if they refer to the object as a whole, and do not operate on parts of the object specified with arguments of this method. This makes names semantically more precise, and we avoid the risk of two method names of two different interfaces at the same object folding into each other causing problems with scripting languages.

Special attention should be given to uniformity within semantically related interfaces. This means, if a method is named `destroyRecord()`, an insert method should be called `insertRecord()`.

If a method refers to a part of the object and an argument specifies this part, the type or role of the part is appended to the verbal part of the method,for example, `removeFrame( [in] XFrame xFrame )`. If the name of the part or its position is specified as an argument, ByName or ByIndex is additionally appended, for example, `removeFrameByName( [in] string aName )` or `removeFrameByIndex( [in] long nIndex )`.

The following method prefixes have special meanings:

get
: To return non-boolean values or interfaces of other objects that have a lasting relationship with the object the associated interface belongs to, similar to being an attribute. This prefix is generated automatically for readable attributes. Multiple calls to the same method at the same object with the same arguments, without modifying the object in between, returns the same value or interface.

set
: To set values or interfaces of other objects that get into a lasting relationship with the object the associated interface belongs to, similar to becoming attribute values. This prefix is generated automatically for writable attributes.

query
This prefix is used to return values, including interfaces that have to be calculated at runtime or do not have the character of being a structural part of the object which belongs to the associated interface. Multiple calls, even without modifying the object in between, do not necessarily return the same value and interface; but this can be specified in the specific methods.

is/has
Usage is similar to get, and is used for boolean values.

create
This prefix is used for factory methods. Factory methods create and return new instances of objects. In many cases, the same or a related interface has an insert or add method.

insert
This prefix inserts new sub objects into an object when the insertion position is specified.

add
This prefix inserts new sub objects into an object when the insertion position is not specified by any argument of the method.

append
This prefix inserts new sub objects into an object when the new sub object gets appended at the end of the collection of sub objects.

remove
This prefix removes sub objects from a container. Use destroy if the removal implies the explicit destruction of the sub object. If the sub object is given as an argument, use it's type or role additionally, for example, `removeFrame()` if the argument is a Frame. If the position index or name of the sub object to remove is given, use a name similar to `removeFrameByName()`. For generic interfaces, use `removeByName()` without the type name or role, which are unknown in that case.

destroy
This prefix removes sub objects from a container and explicitly destroys them in this process. Use destroy as a verbal prefix. For more details, see the description of remove.

clear
This prefix clears contents of an object as a whole, the verb itself, or certain parts of the object, such as add a specifying name giving something like `clearDelegates()`.

dispose
This prefix initiates a broadcast message to related objects to clear references to the object. Normally, this verb is only used in `XComponent`.

approve
This prefix is used for the approval notification in listener interfaces of prohibited events.

Usage

Non-structural attributes are represented by the property concept, and not by get and set methods, or attributes of interfaces.

Consider possible implementations if there are several possible interfaces where you could put a method. For example, a file cannot destroy itself, but the container directory could.

Do not use `const` as an attribute for methods, because future versions of UNOIDL will not support this feature.

A.2.3 Interfaces

Interfaces are collections of methods belonging to a single aspect of behavior. Methods do not have data or implementation.

Once an interface gets into an official release of the API, it may no longer be changed. This means that no methods or attributes can be added, removed or modified, not even arguments of methods. This rule covers syntax, as well as semantics.

Interfaces are identified by their name.

Naming

Identifiers of interfaces begin with the prefix 'X' followed by the name itself in initial caps, capitalizing the first letter after the 'X', for example, XFrameListener. Avoid abbreviations.

We apply the prefix 'X', because interfaces have to be treated differently than pointers in C/C++ and also in normal interfaces in Java. It is also likely that the main interface of a service should get the same name as the service that can cause confusion or ambiguity in documentation.

It is a bad design if the name or abbreviation of a specific component appears within the name of an interface, for example, XSfxFrame or XVclComponent.

Usage

Interfaces represent stable aspects of design objects. A single interface only contains methods that belong to one aspect of object behavior, never collections of arbitrary methods. Both aspects of usage, client and server, should be considered in design. Keep the role of the object in mind. If some methods of your new interface are only used in one role and others in another role, your design is probably flawed.

A.2.4 Properties

Properties are descriptive attributes of an objects that can be queried and changed at runtime using the `XPropertySet` interface.

Naming

In non-scripting languages, such as Java or C/C++, property identifiers are simply strings. These identifiers always begin with an uppercase letter and use initial caps , for example, `BackgroundColor`. Avoid abbreviations.

Usage

Properties are used for non-structural attributes of an object. For structural attributes (composition) use get and set methods in an interface instead.

A.2.5 Events

Events are notifications that you can register as listeners. This concept is actually expressed by registration or unregistration methods for the broadcaster, listener interfaces for the listener and event structures for the event.

Naming

If an object broadcasts a certain event, it offers a pair of methods like `addEventNameListener()` and `removeEventNameListener()`. This scheme conforms to the naming scheme of JavaBeans and does not mean that the implementation keeps track of a separate list for each event.

The event methods of the listener interface use the past tense form of the verb that specifies the event, usually in combination with the subject to which it applies, for example, `mouseDragged()`. For events which are notified before the event actually happens, the method begins with notify, for example, `notifyTermination()`. Event methods for prohibited events start with the prefix approve, for example, `approveTermination()`.

Usage

Use events if other, previously unknown objects have to be notified about status changes in your object.

Normally, the methods `add...Listener()` and `remove...Listener()` have a single argument. The type of argument is an interface derived from `com.sun.star.lang.XEventListener`.

The event is a struct derived from `com.sun.star.lang.EventObject`, therefore this struct contains the source of the event.

A.2.6 Services

Services are collections of related interfaces and properties. They specify the behavior of implementation objects at an abstract level by specifying the relationship and interaction between these interfaces and properties. Like interfaces, services are strictly abstract.

Naming

Service identifiers begin with an uppercase letter and are put in initial caps , for example, `com.sun.star.text.TextDocument`). Avoid abbreviations.

Usage

Services are used by a factory to create objects which fulfill certain requirements. Not all services are able to be instantiated by a factory, but they are used for documentation of properties or interface compositions. In a service, you can specify in detail what methods expect as arguments or what they return.

A.2.7 Exceptions

Exceptions are special classes which describe exceptional states.

Naming

Exception identifiers begin with a capital uppercase letter, and are put in initial caps and always end with Exception, (for example, `com.sun.star.lang.IllegalArgumentException`. Avoid abbreviations.

Usage

The OpenOffice.org API uses exceptions as the general error handling concept. However, the API should be designed that it is possible to avoid exceptions in typical error situations, such as opening non-existent files.

A.2.8 Enums

Enums are non-arbitrary sets of identifying values. If an interface uses an enum type, all implementations have to implement all specified enum values. It is possible to specify exceptions at the interface. Extending enums is not allowed, because this would cause incompatibilities.

Naming

Enum types begin with an uppercase letter and are put in initial caps. Avoid abbreviations. If there is a possible name-conflict with structs using the same name, add Type or Style to the enum identifier.

Enum values are completely capitalized in uppercase and words are separated by underscores. Do not use a variant of the enum type name as a prefix for the values, because some language bindings will do that automatically.

```
enum FooBarType
{
 NONE,
 READ,
 WRITE,
 USER_DEFINED = 255
};

struct FooBar
{
 FooBarType Type;
 string FileName
};
```

Three typical endings of special enum values are `_NONE`, `_ALL` and `_USER_DEFINED`.

Usage

If by general fact an enum represents the most common values within an open set, add a value for USER_DEFINED and specify the actual meaning by a string in the same object or argument list where the enum is used. In this case, offer a method that returns a sequence of all possible values of this string.

A.2.9 Typedefs

Typedefs specify new names for existing types.

Naming

Typedefs begin with an uppercase letter and are put in initial caps. Avoid abbreviations.

Usage

Do not use typedefs in the OpenOffice.org API.

A.2.10 Structs

Structs are static collections of multiple values that belong to a single aspect and could be considered as a single, complex value.

Naming

Structs begin with an uppercase letter and are put in initial caps. Avoid abbreviations.

If the actual name for the struct does not sound correct, do not add `Attributes`, `Properties` or the suffixes suggested for enums. These two words refer to different concepts within the OpenOffice.org API. Instead, use words like `Format` or `Descriptor`.

Usage

Use structs as data containers. Data other than interfaces are always copied by value. This is an efficiency gain, especially in distributed systems.

Structs with just a single member are wrong by definition.

A.2.11 Parameter

Parameters are names for arguments of methods.

Naming

Argument identifiers begin with a special lowercase letter as a prefix and put in initial caps later, for example, `nItemCount`.

Use the following prefixes:

- 'x' for interfaces
- 'b' for boolean values
- 'n' for integer numbers

- 'f' for floating point numbers
- 'a' for all other types. These are represented as classes in programming languages.

Usage

The order of parameters is defined by the following rule: Where, What, How. Within these groups, order by importance. For example, `insertContent(USHORT nPos, XContent xContent,` and `boolean bAllowMultiInsert`.

A.3 Special Cases

Error Handling (Exceptions/Error-Codes)

Runtime errors caused by the wrong usage of interfaces or do not happen regularly, raise exceptions. Runtime errors that happen regularly without a programming mistake, such as the non-existence of a file for a file opening method, should be handled by using error codes as return values.

Collection Interfaces

Collection-Services usually support one or multiple `X...Access-Interfaces` and sometimes add access methods specialized to the content type. For example,

```
XInterface XIndexAccess::getElementByIndex(unsigned short)
```

becomes

```
XField XFields::getFieldByIndex(unsigned short).
```

Postfix Document for Document-like Components

Components, whose instances are called a document, get the postfix `Document` to their name, for example, service `com.sun.star.text.TextDocument`.

Postfixes Start/End vs. Begin/End

The postfixes ...Start/...End are to be preferred over ...Begin/...End.

A.4 Abbreviations

Avoid abbreviations in identifiers of interfaces, services, enums, structs, exceptions and constant groups, as well as identifiers of constants and enum values. Use the following open list of abbreviations if your identifier is longer than 20 characters. Remain consistent in parallel constructions, such as `addSomething()` or `removeSomething()`.

- `Abs`: Absolute
- `Back`: Background

- `Char`: Character
- `Doc`: Document
- `Ext`: Extended, Extension
- `Desc`: Description, Descriptor
- `Ref`: Reference
- `Hori`: Horizontal
- `Orient`: Orientation
- `Para`: Paragraph
- `Var`: Variable
- `Rel`: Relative
- `Vert`: Vertical

A.5 Source Files and Types

For each type, create a separate IDL file with the same base name and the extension *.idl*.

Appendix B: IDL Documentation Guidelines

B.1 Introduction

The reference manual of the OpenOffice.org API is automatically generated by the tool *autodoc* from the idl files that specify all types used in the OpenOffice.org API. These files are part of the SDK. This appendix discusses how documentation comments in the idl files are used to create correct online documentation for the OpenOffice.org API.

B.1.1 Process

The *autodoc* tool evaluates the UNOIDL elements and special JavaDoc-like documentation comments of these files, but not the C++/Java-Style comments. For each element that is documented, the preceding comment is used. Put the comment within /** .... */ for multiple-line documentation, or put behind /// for single-line documentation.



Do not put multiple documentation comments! Only the last will be evaluated for each element and appear in the output.

```
/// This documentation will be lost.
/// And this documentation will be lost too.
/// Only this line will appear in the output!
```

Most XHTML tags can be used within the documentation, that is, only tags that occur between the `<body>...</body>` tags. Additionally, other XML tags are supported and JavaDoc style @-tags can be used. These are introduced later.

It is good practice and thus recommended to build a local version of newly written IDL documentation and browse the files with an HTML client to check if all your layouts appear correctly.

B.1.2 File Assembly

Each individual idl file only contains a single type, that is, a single interface, service, struct, enum, constants group or exception. Nested types are not allowed for OpenOffice.orgs local API., even though they are supported by UNOIDL.

B.1.3 Readable & Editable Structure

The idl files have to be structured for easy reading and re-editing. Indentation or line-breaks are generated automatically in the output of *autodoc*, but the simple ASCII layout of the documentation comments has to be structured for readability of the idl files. Due to HTML interpretation, the line-breaks do not appear in the output, except in `<pre>...</pre>` and similar areas.

The idl files should not contain any #ifdef or #if directives than those mentioned in this document because they are read by the OpenOffice.org community and others. Do not introduce task force or version dependent elements, use CVS branches instead.

Avoid leading asterisks within documentation blocks. Misplaced asterisks must be removed when reformatting is necessary, thus time consuming.



Do not use leading asterisks as shown here:

```
/* does something special.
 *
 * This is just an example for what you must NOT do: Using leading asterisks.
 */
```

B.1.4 Contents

The idl files should not contain implementation specific information. Always consider idl files as part of the SDK delivery, so that they are visible to customers.

B.2 File structure

This chapter provides information about the parts of each idl file, such as the header, body and footer, the character set to be used and the general layout to be applied.

B.2.1 General

Length of Lines

Lines in the idl files should not be longer than 78 characters, and documentation comment lines should not be longer than 75 characters. The preferable length of lines is upto 70 characters. This makes it readable in any ASCII editor and allows slight changes, that is, due to English proof-reading without the need of reformatting.

Character Set and Special Characters

Only 7-bit ASCII characters are used in UNOIDL, even in the documentation comments. If other characters are necessary, the XHTML representation is to be used. See *http://www.w3.org/TR/xhtml1/DTD/xhtml-special.ent* for a list of the encodings.

Completeness of Sentences

In general, build grammatically complete sentences. One exception is the first sentence of an elements documentation, it may begin with a lowercase letter, in which case the described element itself is the implied subject.

Indentation

The indentation of sub-elements and for others is four spaces for each level of indentation.

Delimiters

Each major element has to be delimited by a 75 to 78-character long line from the other major elements. This line consists of “//” followed by equal signs to match the regular expression "^/*=*$". Place it immediately in the line above the documentation comment that it belongs to.

Major elements are typedef, exception, struct, constants, enum, interface and service.

The sub elements can be delimited by a 75 to 78-character long line matched by the regular expression
"^ \(\)*/*- *$" from the other minor elements and the major element. This is a line consisting of a multiple of four spaces, followed by “//” and dashes. Place it immediately in the line above the documentation comment that it belongs to. Minor elements are structure and exception fields, methods and properties. Interfaces and services supported by services as single constants are to be grouped by delimiters.

Examples for major and minor elements are given below.

B.2.2 File-Header

For legal reasons, the header has to be exactly as shown in the following snippet. Exceptions of this rule are the dynamic parts within "$...$" and the list of contributors at the end.

```
/*************************************************************************
 *
 * $RCSfile: IDLDocumentationGuide.xml,v $
 *
 * $Revision: 1.11 $
 *
 * last change: $Author: jsc $ $Date: 2003/05/26 09:39:19 $
 *
 * The Contents of this file are made available subject to the terms of
 * either of the following licenses
 *
 *     - GNU Lesser General Public License Version 2.1
 *     - Sun Industry Standards Source License Version 1.1
 *
 * Sun Microsystems Inc., October, 2000
 *
 *
 * GNU Lesser General Public License Version 2.1
 * =============================================
 * Copyright 2000 by Sun Microsystems, Inc.
 * 901 San Antonio Road, Palo Alto, CA 94303, USA
 *
 * This library is free software; you can redistribute it and/or
 * modify it under the terms of the GNU Lesser General Public
 * License version 2.1, as published by the Free Software Foundation.
 *
 * This library is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
 * Lesser General Public License for more details.
 *
```

```
* You should have received a copy of the GNU Lesser General Public
* License along with this library; if not, write to the Free Software
* Foundation, Inc., 59 Temple Place, Suite 330, Boston,
* MA 02111-1307 USA
*
*
* Sun Industry Standards Source License Version 1.1
* =================================================
* The contents of this file are subject to the Sun Industry Standards
* Source License Version 1.1 (the "License"); You may not use this file
* except in compliance with the License. You may obtain a copy of the
* License at http://www.openoffice.org/license.html.
*
* Software provided under this License is provided on an "AS IS" basis,
* WITHOUT WARRUNTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING,
* WITHOUT LIMITATION, WARRANTIES THAT THE SOFTWARE IS FREE OF DEFECTS,
* MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE, OR NON-INFRINGING.
* See the License for the specific provisions governing your rights and
* obligations concerning the Software.
*
* The Initial Developer of the Original Code is: Sun Microsystems, Inc..
*
* Copyright: 2002 by Sun Microsystems, Inc.
*
* All Rights Reserved.
*
* Contributor(s): _____________________________________
*
*
************************************************************************/
```

The filename in "$RCSfile: IDLDocumentationGuide.xml,v $" is replaced automatically by the version control system, as well as "$Revision: 1.11 $", "$Author: jsc $" and "$Date: 2003/05/26 09:39:19 $". Contributors add their names to the list at the end.

The copyright date has to be adapted to the actual last year of work on the file.

The #ifdef and #define identifiers consist of two underscores "__", the module specification, each nested module separated by a single underscore "_" and the name of the file separated with a single underscore "_" as shown above and trailing two underscores "__".

B.2.3 File-Footer

The files do not have a footer with VCS fields. The history can only be viewed from CVS directly. This is to avoid endless expanding log lists.

B.3 Element Documentation

B.3.1 General Element Documentation

Each element consists of three parts:

1. a summary paragraph with XHTML/XML markups
2. the main description with XHTML/XML markups
3. extra parts formed by @-tags

Summary Paragraph

The first part ending with an XHTML paragraph tag, that is, <p>, <dl>, <ul>, <ol> etc.) or "@..." tag, is used as the summary in indexes.



In contrast to JavaDoc, the first sentence is not used for the summary, but the first paragraph.

The first sentence begins with a lowercase letter if the name of the described element is the implied noun. In this case, the sentence must be logical when reading it with that name. Sometimes an auxiliary verb. in the most cases "is", has to be inserted.

Main Description

Between the summary paragraph and the "@..." tag there should be a clear and complete description about the declared element. This part must be delimited from the summary paragraph with an XHTML paragraph tag, including "<dl>" and "<ol>", that are starting a new paragraph.

@-Tagged Part

Put the @tags at the end of each element's documentation. The tags are dependent on the kind of element described. Each of the @-tag ends when the elements documentation ends or the next @-tag begins.

The @author tag is superfluous, because the author is logged by the version control system. They are only used for OpenOffice.org contributions if declarations are taken from other projects, such as Java.

The @version tag, known from JavaDoc, is not valid, because there cannot be more than one version of any UNOIDL element, due to compatibility.

On the same line behind the @-tag, only a single structural element is allowed. The parameter name is @param without the type and any attributes, the qualified exception is @throws , the qualified type is @see, and the programming language is @example. The @returns is by itself on the same line.



Do not put normal text behind an @-tag on the same line:

```
/** ...

  @param nPos put nothing else but the argument name here!
    it is correct to put your documentation for the parameter here.

  @throws com::sun::star::beans::UnknownPropertyException nothing else here!
    when <var>aName</var> is not a known property.
*/
```

B.3.2 Example for a Major Element Documentation

Each major element gets a header similar to the example shown below for an interface:

```
//=============================================================================
/** controls lifetime of the object. Always needs a specified object owner.

  <p><em>Logical</em> "Object" in this case means that the interfaces
  actually can be supported by internal (i.e. aggregated) physical
  objects. </p>

  @see com::sun::star::uno::XInterface
```

```
    for further information.
*/
interface XComponent: XInterface
{
```

B.3.3 Example for a Minor Element Documentation

Each minor element gets a header similar to the example shown below for a method:

```
    //------------------------------------------------------------------------
    /** adds an event listener to the object.

    <p>The broadcaster fires the disposing method of this listener if
    the <method>XComponent::dispose()</method> method is called. </p>

    @param xListener
      refers the the listener interface to be added.

    @returns
      <TRUE/> if the element is added, <FALSE/> otherwise.

    @see removeEventListener
    */
    boolean addEventListener( [in]XEventListener xListener );
```

B.4 Markups and Tags

B.4.1 Special Markups

These markup tags are evaluated by the XSL processor that generates a layout version of the documentation, that is, into HTML or XHTML. These tags have to be well formed andin pairs with exactly the same upper and lowercase, as well.

To accentuate identifiers in the generated documentation and generate hyperlinks automatically when building the cross-reference table and checking consistency, all identifiers have to be marked up. Additionally, it is important for proofreading, because a single-word method name cannot be distinguished by a verb. Identifiers have to be excluded from re-editing by the proofreading editor.

The following markups are used:

<atom>

This markup is used for identifiers of atomar types, such as long, short, and string. If a sequence or array of the type is referred to, add the attribute dim with the number of bracket-pairs representing the number of dimensions.

Example:

```
<atom>long</atom>
```

For an example of sequences, see <type>.

<type>

This markup is used for identifiers of interfaces, services, typedefs, structs, exceptions, enums and constants-groups. If a sequence or array of the type is referred to, add the attribute dim with the number of bracket-pairs representing the number of dimensions.

Example:

```
<type scope="com::sun::star::uno">XInterface</type>
<type dim="[][]">PropertyValue</type>
```

<member>

This markup substitutes the deprecated method, field and property markups, and is used for fields of structs and exceptions, properties in service specifications and methods of interfaces.

Example:

```
<member scope="com::sun::star::uno">XInterface::queryInterface()</member>
```

<const>

This markup is used for symbolic constant identifiers of constant groups and enums.

Example:

```
<const scope="...">ParagraphAdjust::LEFT</const>
```

<TRUE/>, <FALSE/>

These markups represent the atomic constant for the boolean values TRUE and FALSE.

Example:

```
@returns
<TRUE/> if the number is valid, otherwise <FALSE/>.
```

<NULL/>

This markup represents the atomic constant for a NULL value.

<void/>

This markup represents the atomic type void. This is identical to `<atom>void</atom>`.

<code>

This markup is used for inline code.

Example:

Use <code>queryInterface(NULL)</code> for:

```
<listing>
This markup is used for multiple line code examples.
Example:
```

```
@example StarBASIC
<listing>
aCmp = StarDesktop.loadComponentFromURL( ... )
if ( isnull(aCmp) )
....
endif
</listing>
```

B.4.2 Special Documentation Tags

This group of special tags are analogous to JavaDoc. Only what has previously been mentioned in this guideline can appear in the line behind these tags. The pertaining text is put into the line following . Each text belonging to a tag ends with the beginning of the next special tag ("@") or with the end of the documentation comment.

@author Name of the Author

This tag is only used if an element is adapted from an externally defined element, that is, a Java class or interface. In this case, the original author and the in-house author at Sun Microsystems is mentioned.

Example:

```
@author John Doe
```

@see qualifiedIdentifier

This tag is used for extra cross references to other UNOIDL-specified elements. Some are automatically generated, such as all methods using this element as a parameter or return value, and services implementing an interface or include another service. If there is no other method that should be mentioned or an interface with a similar functionality, it should be referenced by this @see statement.

A @see-line can be followed by further documentation.

Example:

```
@see com::sun::star::uno::XInterface::queryInterface
  For this interface you have always access to ...
```



Do not use markups on the identifier on the same line behind the @see-tag!

```
/** ...

  @see <type>these markups are wrong</type
*/
```

@param ParameterName

This tag describes the formal parameters of a method. It is followed by the exact name of the parameter in the method specification. The parameter by itself may be the implicit subject of the following sentence, if it begins with a lowercase letter.

Examples:

```
@param xListener
  contains the listener interface to be added.
@param aEvent
  Any combination of ... can be used in this parameter.
```

@return/@returns

This tag starts the description of the return value of a method. The description is in the line folloiwng. If it begins with a lowercase letter, the method is the implied subject and "returns" is the implied verb. See the following example:

```
@returns
  an interface of an object which supports
  the <type>Foo</type> service.
```

@throws qualifiedException

This tag starts the description of an exception thrown by a method in a particular case. The exception type is stated behind in the same line and must be fully qualified, if it is not in the same module. The description is in the line following. If it begins with a lowercase letter, the method is the implied subject and "throws" is the implied verb.

Example:

```
@throws com::sun::star::uno::lang::InvalidArgumentException
  if the specified number is not a specified ID.
```

@version VersionNumber

This was originally used to set a version number for the element. This tag is deprecated and should not be used.

B.4.3 Useful XHTML Tags

Only a few XHTML tags are required for writing the documentation in idl files. The most important ones are listed in this section.

Paragraph: ...

This tag marks a normal paragraph. Consider that line breaks and empty lines in the idl file do not cause a line break or a paragraph break in the layout version. Explicit paragraph break markups, are necessary.



Do not use
 or CR/LF for marking paragraphs. CR and LF are ignored, except within <pre>...</pre> and <listing>...</listing> areas. The
 tag is only for rare cases of explicit linebreaks.

```
/** does this and that.

  This sentence should start with a "&lt;p&gt;". If not,
  it still belongs to the previous paragraph!

  This still belongs to the first paragraph. <br/>
  As this sentence is as well!
*/
```

Consider using < for < and > for >, as shown in the example above.

Line Break:

This tag marks up a line break within the same paragraph. Consider line breaks and empty lines in the idl file do not cause a line break or a paragraph break when presented by the HTML browser. Explicit paragraph break markups are necessary.

Unordered List: ... ...

These tags mark the beginning and end of an unordered list, as list items.

Example:

```
<ul>
  <li>the first item </li>
  <li>the second item </li>
  <li>the third item </li>
</ul>
```

results in a list similar to:

- the first item
- the second item
- the third item

Ordered List: ... ...

These tags mark the beginning and end of an ordered list, as list items.

Example:

```
<ol>
  <li>the first item </li>
  <li>the second item </li>
  <li>the third item </li>
</ol>
```

results in a list similar to:

1.the first item

2.the second item

3.the third item

Definition List: <dl><dt> ... </dt><dd> ... </dd>... </dl>

These tags mark the beginning and end of a definition list, the definition tags and the definition data.

Example:

```
<dl>
  <dt>the first item</dt>
  <dd>asfd asdf asdf asdf asdf</dd>

  <dt>the second item</dt>
  <dd>asfd asdf asdf asdf asdf</dd>

  <dt>the third item</dt>
  <dd>asfd asdf asdf asdf asdf</dd>
</dl>
```

results in a list similar to:

the first item

asfd asdf asdf asdf asdf

the second item

asfd asdf asdf asdf asdf

the third item

asfd asdf asdf asdf asdf

Table: <table><tr><td>...</td>...</tr>...</table>

Defines a table with rows (tr) and columns (td).

Strong Emphasis: ...

These tags present a piece of text that is emphasized. In most cases this is bold, but the HTML-client defines what it actually is.

Slight Emphasis: ...

These tags present a piece of text emphasized slightly. In most cases this is italic, but the HTML-client defines what it actually is .

Anchor: ...

These tags specify a link to external documentation. The first "..." specifies the URL.

Appendix C: Universal Content Providers

C.1 The Hierarchy Content Provider

C.1.1 Preface

The Hierarchy Content Provider (HCP) implements a content provider for the Universal Content Broker (UCB). It provides access to a persistent, customizable hierarchy of contents.

C.1.2 HCP Contents

The HCP provides three different types of contents: *link*, *folder* and *root folder*.

1. An HCP link is a content that "points" to another UCB content. It is always contained in an HCP Folder. An HCP Link has no children.
2. An HCP folder is a container for other HCP Folders and HCP Links.
3. There is at least one instance of an HCP root folder at a time. All other HCP contents are children of this folder. The HCP root folder contains HCP folders and links. It has the URL *vnd.sun.star.hier:/*.

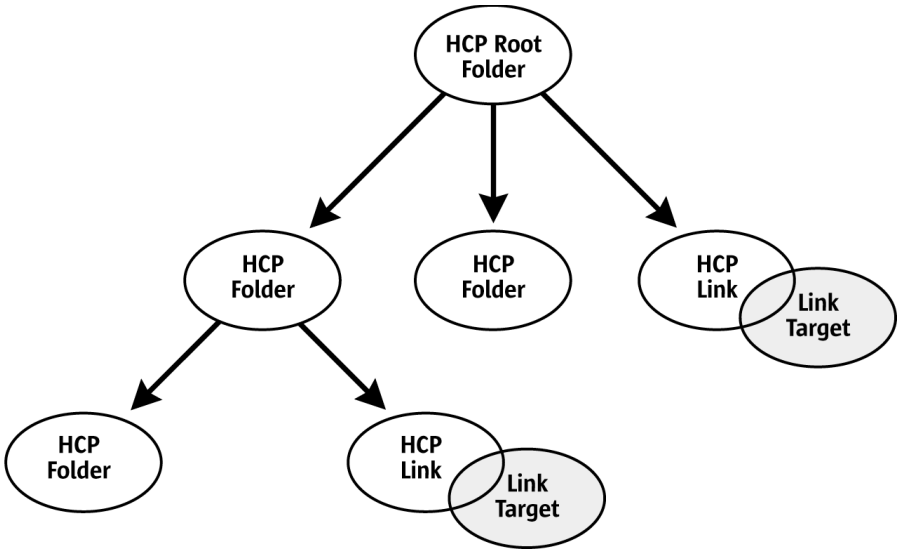


Illustration 213

C.1.3 Creation of New HCP Content

HCP folders and the HCP root folder implement the interface `com.sun.star.ucb.XContentCreator`. HCP links and HCP folders support the command `"insert"` allowing all the HCP folders, as well as the HCP root folder to create new HCP folders and HCP links. To create a new child of an HCP folder:

1. The parent folder creates a new content by calling its `createNewContent()` method. The content type for new folders is "application/vnd.sun.star.hier-folder". To create a new link, use the type string "application/vnd.sun.star.hier-link".
2. Set a title at the new folder or link. The new child executes the `"setPropertyValues"` command that sets the property `Title` to a non-empty value. For a link, set the property `TargetURL` to a non-empty value.
3. The new child, not the parent executes the command `"insert"`. This commits the creation process.

C.1.4 URL Scheme for HCP Contents

Each HCP content has an identifier corresponding to the following scheme:

vnd.sun.star.hier:/<path>

where *<path>* is a hierarchical path of the form

name>/<name>/.../<name>

where *<name>* is an encoded string according to the URL conventions.

Examples:

vnd.sun.star.hier:/ (The URL of the HCP Root Folder)

vnd.sun.star.hier:/Bookmarks/Sun%20Microssystems%20Home%20Page

vnd.sun.star.hier:/Workplace/Printers

C.1.5 Commands and Properties

	UCB Type (returned by XContent::getContentType)	Properties	Commands	Interfaces
Link	application/ vnd.sun.star.hier-link	[readonly] ContentType [readonly] IsDocument [readonly] IsFolder Title TargetURL	getCommandInfo getPropertySetInfo getPropertyValues setPropertyValues insert delete	`XTypeProvider` `XServiceInfo` `XComponent` `XContent` `XCommandProcessor` `XPropertiesChan-geNotifier` `XPropertyContainer` `XPropertySetInfo-ChangeNotifier` `XCommandInfoChan-geNotifier` `XChild`
Folder	application/ vnd.sun.star.hier-folder	[readonly] ContentType [readonly] IsDocument [readonly] IsFolder Title	getCommandInfo getPropertySetInfo getPropertyValues setPropertyValues insert delete open transfer[1]	same as HCP Link, plus `XContentCreator`
Root Folder	application/ vnd.sun.star.hier-folder	[readonly] ContentType [readonly] IsDocument [readonly] IsFolder Title	getCommandInfo getPropertySetInfo getPropertyValues setPropertyValues open transfer	same as HCP Link, plus `XContentCreator`

1 The "`transfer`" command only transfers HCP-contents to HCP folders. It does not handle contents with a URL scheme other then the HCP-URL-scheme.

C.2 The File Content Provider

C.2.1 Preface

The File Content Provider (FCP), a content provider for the Universal Content Broker (UCB), provides access to the local file system by providing file content objects that represent a directory or a file in the local file system. The FCP is able to restrict access to the file system to a number of directories shown to the client under configurable aliases.

C.2.2 File Contents

The FCP provides content representing a *directory* or *file* in the local file system.

1. A directory contains other directories or files.

2. A file is a container for document data or content. The FCP can not determine the `MediaType` property of a file content.

C.2.3 Creation of New File Contents

A content representing directories implements the interface `com.sun.star.ucb.XContentCreator`. A file content object supports the command `"insert"`. To create a new directory or file in a directory:

1. The parent directory creates a new content by calling its `createNewContent()` method. The content type for new folders is "application/vnd.sun.staroffice.fsys-folder". To create a new file, use the type string "application/vnd.sun.staroffice.fsys-file". A new file content object is the retun value.
2. Set a title at the new file content object. The new child executes a `"setPropertyValues"` command that sets the property `Title` to a non-empty value.
3. The new file content object, not the parent, executes the command `"insert"`. This creates the corresponding physical file or directory. For files, supply the implementation of an `com.sun.star.io.XInputStream` with the command's parameters that provide access to the stream data.

C.2.4 URL Schemes for File Contents

The file URL Scheme

Each file content has an identifier corresponding to the following scheme:

file:///<path>

where *<path>* is a hierarchical path of the form

<name1>/<name>/.../<name>.

The first part of *<path> (<name1>)* is not required to denote a physically existing directory**,** but may be remapped to such a directory. If this is done, the FCP refuses file access for any URL whose <name1>-part is not an element of a predefined list of alias names.

The vnd.sun.star.wfs URL Scheme

In the Sun ONE Webtop, the server-side file system is addressed with vnd.sun.star.wfs URLs. The wfs stands for *Webtop File System*. The file URL scheme is reserved for a potential client-side file system.

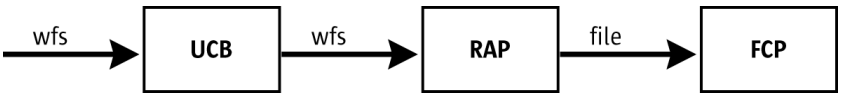


Illustration 214

The vnd.sun.star.wfs URL scheme is completely hidden from the FCP, that is, the server side FCP internally works with file URLs, like any other FCP: There is a Remote Access Content Provider (RAP) between the UCB and the FCP. The RAP, among other things, can route requests to another

UCP and rewrite URLs. This feature is used so that the client of the UCB works with vnd.sun.star.wfs URLs and the FCP remains unmodified and works with file URLs, with a RAP in between that maps between those two URL schemes.

Except for the different scheme name, the syntax of the vnd.sun.star.wfs URL scheme is exactly the same as the file URL scheme.

C.2.5 Commands and Properties

	UCB Type (returned by XContent::getContentType)	Properties	Commands	Interfaces
File	application/ vnd.sun.staroffice.fsys-file	[readonly] ContentType DateCreated DateModified [readonly] IsDocument [readonly] IsFolder Size Title IsReadOnly	getCommandInfo getPropertySetInfo getPropertyValues setPropertyValues insert delete open transfer	XServiceInfo XComponent XContent XCommandProcessor XPropertiesChangeNotifier XPropertyContainer XPropertySetInfoChangeNotifier XCommandInfoChangeNotifier XChild XContentCreator
Directory	application/ vnd.sun.staroffice.fsys-folder	[readonly] ContentType DateCreated DateModified [readonly] IsDocument [readonly] IsFolder Size Title IsReadOnly	getCommandInfo getPropertySetInfo getPropertyValues setPropertyValues insert delete open	XServiceInfo XComponent XContent XCommandProcessor XPropertiesChangeNotifier XPropertyContainer XPropertySetInfoChangeNotifier XCommandInfoChangeNotifier XChild

C.3 The FTP Content Provider

C.3.1 Preface

The FTP content provider implements a content provider for the Universal Content Broker (UCB). It provides access to the contents, folders and documents, made available by FTP servers.

C.3.2 FTP Contents

The FTP Content Provider provides three different types of contents: *accounts*, *folders* and *documents*.

1. An FTP account is a content that represents an account for an FTP server. An account is uniquely determined by a combination of a user name and the host name of the FTP server.

Anonymous FTP accounts have the string "anonymous" as a user name. An FTP account also represents the base directory, that is, the directory that is selected when the user logs in to the FTP server, and behaves like an FTP folder.

2. An FTP folder is a content that represents a directory on an FTP server. An FTP folder never has a content stream, but it can have FTP folders and FTP documents as children.
3. An FTP document is a content that represents a single file on an FTP server. An FTP document always has a content stream and never has children.

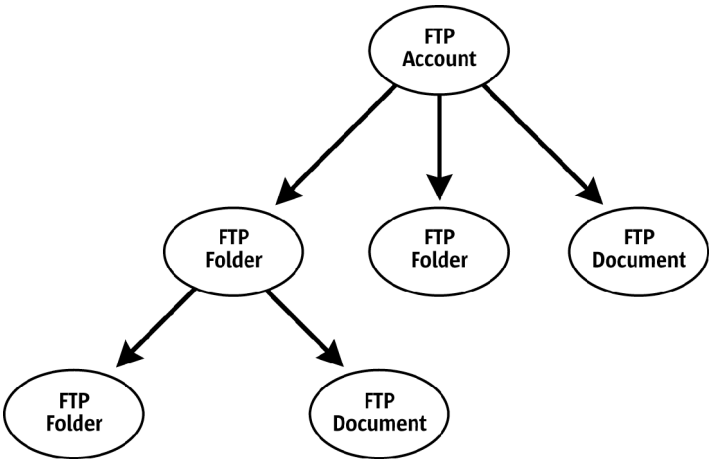


Illustration 215

C.3.3 Creation of New FTP Content

FTP accounts and FTP folders implement the interface `com.sun.star.ucb.XContentCreator`. FTP folders and FTP documents support the command `"insert"`allowing all the FTP accounts and FTP folders to create new FTP folders and FTP documents. To create a new child of an FTP account or FTP folder:

1. The folder creates a new content by calling its `createNewContent()` method. The content type for new folders is "application/vnd.sun.staroffice.ftp-folder". To create a new document, use the type string "application/vnd.sun.staroffice.ftp-file".
2. Set a title at the new folder or document. The new child executes a `"setPropertyValues"` command that sets the property `Title` to a non-empty value.
3. The new child, not the parent, executes the command `"insert"`. This commits the creation process. For documents, supply an `com.sun.star.io.XInputStream`, whose contents are transferred to the FTP server with the command's parameters.

FTP accounts cannot be created the way new FTP folders or FTP documents are created. When you call the FTP content provider's `queryContent()` method with the URL of an FTP account, a content object representing that account, user name and host combination, is automatically created. The same as the URL of an already existing FTP folder or FTP document.

C.3.4 URL Scheme for FTP Contents

Each FTP content has an identifier corresponding to the following scheme (see also RFCs 1738, 2234, 2396, and 2732):

```
ftp-URL ::=             "ftp://" login *("/" segment)
login ::=               [user [":" password] "@"] hostport
user ::=                *(escaped / unreserved / "$" / "&" / "+" / "," / ";" / "=")
password ::=            *(escaped / unreserved / "$" / "&" / "+" / "," / ";" / "=")
hostport ::=            host [":" port]
host ::=                incomplete-hostname / hostname / IPv4address
incomplete-hostname ::= *(domainlabel ".") domainlabel
hostname ::=            *(domainlabel ".") toplabel ["."]
domainlabel ::=         alphanum [*(alphanum / "-") alphanum]
toplabel ::=            ALPHA [*(alphanum / "-") alphanum]
IPv4address ::=         1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
port ::=                *DIGIT
segment ::=             *pchar
pchar ::=               escaped / unreserved / "$" / "&" / "+" / "," / ":" / "=" / "@"
escaped ::=             "%" HEXDIG HEXDIG
unreserved ::=          alphanum / mark
alphanum ::=            ALPHA / DIGIT
mark ::=                "!" / "'" / "(" / ")" / "*" / "-" / "." / "_" / "~"
```

FTP accounts have a login part, optionally followed by a single slash, and no segments. FTP folders have a login part followed by one or more nonempty segments that *must be followed by a slash***.** FTP documents have a login part followed by one or more nonempty segments that *must not be followed by a slash,* that is, the FTP content provider uses a potential final slash of a URL to distinguish between folders and documents. Note that this is subject to change in future versions of the provider.

Examples:

ftp://me@ftp.host
The account of user "me" on the FTP server "ftp.host".

ftp://ftp.host/pub/doc1.txt
A document on an anonymous FTP account.

ftp://me:secret@ftp.host/secret-documents/
A folder within the account of user "me" with the password specified directly in the URL. Not recommended.

C.3.5 Commands and Properties

	UCB Type (returned by XContent::getContentType)	**Properties**	**Commands**	**Interfaces**
Account	application/ vnd.sun.staroffice.ftp-box	[readonly] ContentType [readonly] IsDocument [readonly] IsFolder Title UserName Password FTPAccount[1] ServerName ServerBase[2] [readonly] DateCreated [readonly] DateModified [readonly] FolderCount [readonly] DocumentCount	getCommandInfo getPropertySetInfo getPropertyValues setPropertyValues open transfer[3]	XTypeProvider XServiceInfo XComponent XContent XCommandProcessor XPropertiesChangeNotifier XPropertyContainer XPropertySetInfoChangeNotifier XCommandInfoChangeNotifier XContentCreator
Folder	application/ vnd.sun.staroffice.ftp-folder	[readonly] ContentType [readonly] IsDocument [readonly] IsFolder Title [readonly] DateCreated [readonly] DateModified [readonly] FolderCount [readonly] DocumentCount	getCommandInfo getPropertySetInfo getPropertyValues setPropertyValues insert delete open transfer	same as FTP Account plus XChild

	UCB Type (returned by XContent::getContentType)	Properties	Commands	Interfaces
Document	application/ vnd.sun.staroffice.ftp-file	[readonly] ContentType [readonly] IsDocument [readonly] IsFolder Title [readonly] DateCreated [readonly] DateModified [readonly] IsReadOnly [readonly] Size MediaType	getCommandInfo getPropertySetInfo getPropertyValues setPropertyValues insert delete open	same as FTP Folder minus `XContentCreator`

1 The property `FTPAccount` is similar to `Password`. Some FTP servers not only require authentication through a password, but also through a second token called an "account".

2 The property `ServerBase` is used to override the default directory associated with an FTP account.

3 The "`transfer`" command only transfers contents within one FTP Account. It cannot transfer contents between different FTP accounts or between the FTP content provider and another content provider.

C.4 The WebDAV Content Provider

C.4.1 Preface

The WebDAV Content Provider (DCP) implements a content provider for the Universal Content Broker (UCB). An overview is provided at URLs *http://www.webdav.org* and *http://www.fileangel.org/docs/DAV_2min.html*. It provides access to WebDAV and standard HTTP servers. The DCP communicates with the server by using the WebDAV protocol that is an extension to the HTTP protocol, or by using the plain HTTP protocol if the server is not WebDAV-enabled.

C.4.2 DCP Contents

The DCP provides two types of content: a *folder* or *document* that corresponds to a collection or non-collection, of nodes and leafs, in WebDAV, respectively.

1. A DCP folder is a container for other DCP Folders or Documents.
2. A DCP document is a container for document data or content. The data or content can be any type. A WebDAV server, like an HTTP server, does not mandate what type of data or content is contained within Documents. The type of data or content is defined by the `MediaType` property which is different from the content type returned from the `getContentType()` method. The `MediaType` property is mapped to the equivalent WebDAV property and the WebDAV server calculates the value.

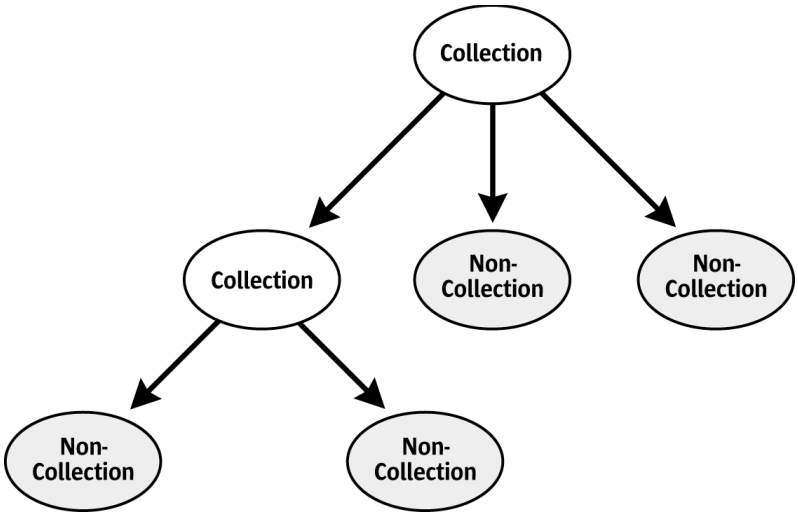


Illustration 216

C.4.3 Creation of New DCP Contents

DCP folders implement the interface `com.sun.star.ucb.XContentCreator`. DCP documents and DCP folders support the command `"insert"`. To create a new child of a DCP folder:

1. The parent folder creates a new content by calling its `createNewContent()` method. The content type for new folders is "application/vnd.sun.star.webdav-collection". To create a new document, use the type string "application/http-content".
2. Set a title for the new folder or document. The new child executes a `"setPropertyValues"` command that sets the property `Title` to a non-empty value.
3. The new child, not the parent, executes the command `"insert"`. This commits the creation process and makes the newly-created content on the WebDAV server persistent.

C.4.4 Authentication

DAV resources that require authentication are accessed using the interaction handler mechanism of the UCB. The DAV content calls an interaction handler supplied by the client to let it handle an authentication request. The implementation of the interaction handler collects the user name or password from a location, for example, a login dialog, and supplies this data as an interaction response.

C.4.5 Property Handling

In addition to the mandatory UCB properties, the DCP supports reading and writing all DAV *live* and *dead* properties. Some DAV live properties are mapped in addition to the UCB properties and conversely, that is, *DAV:creationdate* is mapped to `DateCreated`. Adding and removing dead properties is also supported by the implementation of the `XPropertyContainer` interface of a DCP content.

Property Values:

The DCP cannot determine the semantics of unknown properties, thus the values of such properties will always be presented as plain text, as they were returned from the server.

Namespaces:

The following namespaces are known to the DCP:

- DAV:
- *http://apache.org/dav/props/*

Properties with these namespaces are addressed using a UCB property name which is the concatenation of namespace and name, that is, DAV:getcontentlength.

Dead properties with namespaces that are not well-known to the DCP are addressed using a special UCB property name string, that contains both the namespace and the property name. A special property name string must be similar to the following:

```
<prop:the_propname xmlns:prop="the_namespace">
```

The DCP internally applies the namespace *http://ucb.openoffice.org/dav/props/* to all UCB property names that:

- are not predefined by the UCB API.
- do not start with a well-known namespace.
- do not use the special property name string to encode namespace and name.

For example, a client does an `addProperty( .... "MyAdditionalProperty" ... )`. The resulting DAV property has the name `MyAdditionalProperty`, its namespace is *http://ucb.openoffice.org/dav/props/*. However, the DCP client never sees that namespace, but the client can always use the simple name `MyAdditionalProperty`.

DAV / UCB Property Mapping:

DAV:creationdate	DateCreated
DAV:getlastmodified	DateModified
DAV:getcontenttype	MediaType
DAV:getcontentlength	Size
DAV:resourcetype	(used to set IsFolder, IsDocument, ContentType)

C.4.6 URL Scheme for DCP Contents

Each DCP content has an identifier corresponding to the following scheme:

vnd.sun.star.webdav://host:port/<path>

where *<path>* is a hierarchical path of the form

<name>/<name>/.../<name>

where *<name>* is an encoded string according to the URL conventions.

It is also possible to use standard HTTP URLs. The implementation determines if the requested resource is DAV enabled.

Examples:

vnd.sun.star.webdav://localhost/davhome/
vnd.sun.star.webdav://davserver.com/Documents/report.sdw
http://davserver.com/Documents/report.sdw



Note that the WebDAV URL namespace model is the same as the HTTP URL namespace model.

C.4.7 Commands and Properties

	UCB Type (returned by XContent::getContentType)	Properties	Commands	Interfaces
Document	application/ http-content	[readonly] ContentType [readonly] DateCreated [readonly] DateModified [readonly] IsDocument [readonly] IsFolder [readonly] MediaType [readonly] Size 'Title'	getCommandInfo getPropertySetInfo getPropertyValues setPropertyValues insert delete open	XTypeProvider XServiceInfo XComponent XContent XCommandProcessor XPropertiesChangeNotifier XPropertyContainer XPropertySetInfoChangeNotifier XCommandInfoChangeNotifier XChild
Folder	application/ vnd.sun.star.webdav-collection	[readonly] ContentType [readonly] DateCreated [readonly] DateModified [readonly] IsDocument [readonly] IsFolder [readonly] MediaType [readonly] Size Title	getCommandInfo getPropertySetInfo getPropertyValues setPropertyValues insert delete open "transfer	same as DCP Folder, plus XContentCreator

C.5 The Package Content Provider

C.5.1 Preface

The Package Content Provider (PCP) implements a content provider for the Universal Content Broker (UCB). It provides access to the content of ZIP and JAR archive files. It maybe extended to support other packages, such as OLE storages, in the future.

C.5.2 PCP Contents

The PCP provides two different types of contents: *stream* and *folder*.

1. A PCP stream is a content that represents a file inside a package. It is always contained in a PCP folder. A PCP stream has no children.
2. A PCP folder is a container for other PCP folders and PCP streams.

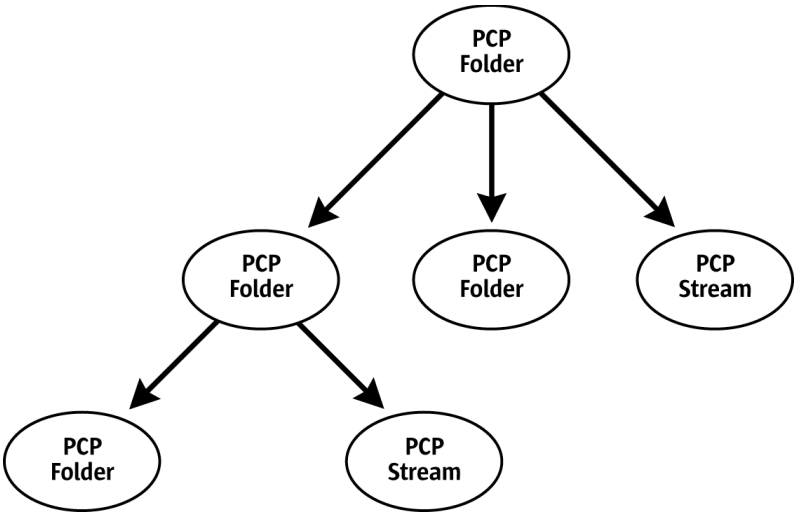


Illustration 217

C.5.3 Creation of New PCP Contents

PCP folders implement the interface `com.sun.star.ucb.XContentCreator`. PCP streams and PCP folders support the command `"insert"`, therefore all PCP folders can create new PCP folders and PCP streams. To create a new child of a PCP folder:

1. The parent folder creates a new content by calling its `createNewContent()` method. The content type for new folders is "application/vnd.sun.star.pkg-folder". To create a new stream, use the type string "application/vnd.sun.star.pkg-stream".
2. Set a title for the new folder or stream. The new child executes a `"setPropertyValues"` command that sets the property `Title` to a non-empty value.
3. The new child, not the parent, executes the command `"insert"`. This commits the creation process. For streams, supply the implementation of an `com.sun.star.io.XInputStream` with the command parameters that provide access to the stream data.

Another convenient method to create streams is to assemble the URL for the new content where the last part of the path becomes the title of the new stream and obtain a Content object for that URL from the UCB. Then, let the content execute the command `"insert"`. The command fails if you set the command parameter "ReplaceExisting" to false and there is already a stream with the title given by the content's URL.

C.5.4 URL Scheme for PCP Contents

Each PCP content has an identifier corresponding to the following scheme:

```
package-URL ::=   "vnd.sun.star.pkg://" orig-URL [ abs-path ]
abs-path ::=      "/" path-segments
path-segments ::= segment * ( "/" segment )
segment ::=       pchar
pchar ::=         unreserved | escaped | ":" | "@" | "&" | "=" | "+" | "$" | ","
unreserved ::=    alphanum | mark
mark ::=          "-" | "_" | "." | "!" | "~" | "*" | "'" | "(" | ")"
escaped ::=       "%" hex hex
orig-URL 1 ::=     * ( unreserved | escaped | "$" | "," | ";" | ":" | "@" | "&" | "&" | "=" | "+" )
```

Examples:

vnd.sun.star.pkg://file:%2F%2F%2Fe:%2Fmy.xsw/
The root folder of the package located at *file:///e:/my.xsw*.

vnd.sun.star.pkg://file:%2F%2F%2Fe:%2Fmy.xsw/Content
The folder or stream named "Content" that is contained in the root folder of the package located at *file:///e:/my.xsw*.

vnd.sun.star.pkg://file:%2F%2F%2Fe:%2Fmy.xsw/Content%20A
The folder or stream named "Content A" that is contained in the root folder of the package located at *file:///e:/my.xsw*.

C.5.5 Commands and Properties

	UCB Type (returned by XContent::getContentType)	**Properties**	**Commands**	**Interfaces**
Stream	application/ vnd.sun.star.pkg-stream	[readonly] ContentType [readonly] IsDocument [readonly] IsFolder MediaType [readonly] Size Title Compressed[1]	getCommandInfo getPropertySetInfo getPropertyValues setPropertyValues insert delete open	`XTypeProvider` `XServiceInfo` `XComponent` `XContent` `XCommandProcessor` `XPropertiesChangeNotifier` `XPropertyContainer` `XPropertySetInfoChangeNotifier` `XCommandInfoChangeNotifier` `XChild`
Folder	application/ vnd.sun.star.pkg-folder	[readonly] ContentType [readonly] IsDocument [readonly] IsFolder MediaType [readonly] Size Title	getCommandInfo getPropertySetInfo getPropertyValues setPropertyValues insert delete open transfer[2] flush[3]	same as PCP Stream, plus `XContentCreator`

1 The property `Compressed` is introduced by package streams to explicitly state if you want a stream to be compressed or not. The default value of this property is determined according to the value suggested by the underlying packager implementation.

2 The "`transfer`" command only transfers PCP folders or streams to other PCP folders. It does not handle contents with a URL scheme other then the PCP-URL scheme.

3 'flush' is a command introduced by the PCP Folder. It takes a void argument and returns void. This command is used to write unsaved changes to the underlying package file. Note that in the current implementation, PCP contents **never flush automatically**! Operations which require a flush to become persistent are: "`setPropertyValues(Title | MediaType)`", "`delete`", "`insert`".

C.6 The Help Content Provider

C.6.1 Preface

Currently, the Help Content Provider has the following responsibilities:

1. It is the interface to a search engine that allows a full-text search, including searching specific scopes, such as headers. The range of accessible scopes depends on the indexing process and is currently not changeable after setup.
2. It delivers a keyword index to its clients.
3. The actual helpcontent has media type "text/html," with some images of type "image/gif" embedded. The content is generated from packed xml files that have to be transformed according to a xsl-transformation to produce the resulting XHTML. There is a cascading style sheet used for formatting the XHTML files of media type "text/css".
4. (It provides its clients the modules for which help is available.

C.6.2 Help Content Provider Contents

The responsibilities mentioned above are fulfilled by providing different kinds of content objects to the client, namely:

- a root content giving general information about the installed help files
- a module content serving as the interface to all search functionality
- picture and XHTML Contents providing the actual content of the transformed help files and pictures

These contents are described below.

C.6.3 URL Scheme for Help Contents

Each Help content has an identifier corresponding to the following scheme:

```
URL ::=          scheme delimiter path? query? anchor?
scheme ::=       "vnd.sun.star.help"
delimiter ::=    "://" | ":/"
path ::=         module ( "/" id )?
query ::=        "?" key-value-list?
keyvaluelist ::= keyvalue ( "&" keyvalue )?
keyvalue ::=     key "=" value
anchor ::=       "#" anchor-name
```

Currently, to have a fault-tolerant system, some enveloping set of this is allowed, but without carrying more information.

Examples:

vnd.sun.star.help://?System=WIN&Language=de

vnd.sun.star.help://swriter?System=WIN&Language=de&Query=text&HitCount=120&Scope=Heading

vnd.sun.star.help://portal/4711?System=UNIX&Language=en-US&HelpPrefix=http%3A%2F%2Fportal%2Fportal

Some information must be given in every URL, namely the value of the keys "System"/"Language."

"System" may be one of "UNIX," "WIN," "OS2" and "MAC". This key parameterizes the XSL transformation applied to the help filesand their content changes according to this parameter, and is mandatory only for helpcontents delivering XHTML-files. This may change in the future.

"Language" is coded as ISO639 language code, optionally followed by "-" and ISO3166 country code. Only the language code part of "Language" is used and directly determines the directory, which is relative to the installation path where the help files are found.

In the following, the term "help-directory" is used to determine the directory named "help" in the office/portal installation. The term "help-installation-directory" is used to denote the particular language-dependent subdirectory of the help-directory that contains the actual help files as a packed jar file, the index, the config files and some other items not directly used by the help content provider.

C.6.4 Properties and Commands

Every creatable content can execute the following commands. It isnot constrained to a particular URL-scheme:

```
com::sun::star::ucb::XCommandInfo getCommandInfo()
com::sun::star::beans::XPropertySetInfo getPropertySetInfo()
com::sun::star::sdbc::XRow getPropertyValues([in] sequence< com::sun::star::beans::Property > )
void setPropertyValues([in] sequence< com::sun::star::beans::PropertyValue > )
```

These commands repeat the information given in the following. The available properties and commands are explained by the following URL examples:

Root Content

vnd.sun.star.help://?System=WIN&Language=de

Properties:

Name	Type	value
'Title'	string	"root"
'IsFolder'	boolean	true
'IsDocument'	boolean	true
'ContentType'	string	"application/vnd.sun.star.help"
'MediaType'	string	"text/css"

Commands:

Return Type	Name	Argument Type
XDynamicResultSet	open[1]	OpenCommandArgument2

1 Return value of this command contains the modules available in the particular installation for the requested language. The modules are currently determined by looking for the files in the help-installation-directory matching "*.db", with the exception of the file "picture.db".

Generally, a module corresponds to a particular application, namely, if the writer application is installed, there should be a module "vnd.sun.star.help://swriter" and so forth.

The written `XOutputStream` or the set `XInputStream` (set at `XActiveDataSink`) makes the cascading style sheet available, which is used to format the XHTML files. Physically, the stream contains the content of the file *custom.css* in the help-directory of the office or portal installation.

Only the key "Language" is used. Any other key may be set, but is ignored.

Module Content

vnd.sun.star.help://swriter?System=WIN&Language=de&Query=text&HitCount=120&Scope=
Heading

Properties:

Name	Type	value
Title	string	determined from config file in help-installation-directory
IsFolder	boolean	true
IsDocument	boolean	false
ContentType	string	"application/vnd.sun.star.help"
KeyWordList	sequence< string >	(See below)
KeyWordRef	sequence< sequence < string > >	(See below)
KeyWordAnchorForRef	sequence< sequence < string > >	(See below)
KeyWordTitleForRef	sequence< sequence < string > >	(See below)
SearchScopes	sequence< string >	(See below)

The help files contain specially marked keywords. The alphabetically sorted list of keywords is contained in the property `KeywordList`.

For example, you are looking for keyword `KeywordList[i]`, where `i` is an integer. The help module in which this keyword should be found is determined by the module part of the content URL, "swriter" in our example. Now `KeywordRef[i]` contains a list of document ids, where the document belonging to that id contains the keyword string `"docid = KeywordRef[i][j]"`.

The location in the XHTML document where this particular keyword refers to is marked by an HTML anchor element:

```
<A name="anchor"/>
```

Here the `anchor` is given by the string `"anchor = KeywordAnchorForRef[i][j]"`.

For our example, the URL of the j document in the swriter module containing the keyword `Keyword[i]` is determined as *vnd.sun.star.help://swriter/docid?System=WIN&Language=de#anchor*.

The keys "HitCount", "Query" and "Scope" have no value with regards to the keywords.

The title of the resulting document is determined directly without having to instantiate the content by the value of `"KeywordTitleForRef[i][j]"`.

The module contents are also the interface to the search engine. A number of additional keys in the URL are necessary, namely the keys:

- HitCount
- Query
- Scope

The value of `Scope` should be one of the strings given by the property `SearchScopes`, currently `"Heading"` or `"FullText"`. Leaving the key undefined defaults to a full-text search, Setting it to `"Heading"` means to search in only the document titles.

There may be any number of `Query` key definitions in the URL. Many `Query` keys determine a query search, first for documents containing all the values, then searching for those containing only subsets of all the values. The requested number of results is determined by the value of the key `HitCount`. The actual returned number may be smaller. The interface to the results returned

by the search engine is given by an `com.sun.star.ucb.XDynamicResultSet`, which is the return value of the command `"open"`:

Return Type	Name	Argument Type
XDynamicResultSet	open	OpenCommandArgument2

XHTML Content or Picture Content

vnd.sun.star.help://portal/4711?System=UNIX&Language=en-US&HelpPrefix=http%3A%2F%2Fportal%2Fportal%2F

Properties:

Name	Type	value
Title	string	determined from database
IsFolder	boolean	false
IsDocument	boolean	true
ContentType	string	"application/vnd.sun.star.help"
MediaType[1]	string	"text/html" or "image/gif"

1 The MediaType "image/gif" corresponds to a URL which contains a module part "picture", as opposed to "portal" in the example.

Commands:

Return Type	Name	Argument Type
void	"open"[1]	OpenCommandArgument2

1 Returns the transformed XHTML-content, or the gif-content of a `PictureContent`.

There is one special document for every module, namely those named *start* (replace *4711* by *start* in our example). It identifies the main help page for every module.

There is an additional key named `HelpPrefix`. If set, every link in a generated document pointing to another help-document, either XHTML or image document, is first encoded and then prefixed by the URL-decoded form of the value of this key. This key is only used by Sun One Webtop.

Appendix D: UNOIDL Syntax Specification

The following listing comprises the language specification for UNOIDL in pseudo BNF notation.

```
(1) <idl_specification> := <definition>+

(2) <definition> := <type_decl> ";"
                  | <module_decl> ";"
                  | <constant_decl> ";"
                  | <exception_decl> ";"
                  | <constants_decl> ";"
                  | <service_decl> ";"
                  | <singleton_decl> ";"

(3) <type_decl> := <interface>
                 | < constr_type_spec >
                 | "typedef" <type_spec> <declarator> {"," <declarator> }*

(4) <interface> := <interface_decl>
                 | <forward_decl>

(5) <forward_decl> := "interface" <identifier>

(6) <interface_decl> := <interface_header> "{" <interface_body> "}"

(7) <interface_header> := "interface" <identifier> [ <interface_inheritance> ]

(8) <interface_inheritance> := ":" <interface_name>

(9) <interface_name> := <scoped_name>

(10)<scoped_name> := <identifier>
                   | "::" <scoped_name>
                   | <scoped_name> "::" <identifier>

(11)<interface_body> := <export>+

(12)<export> := <attribute_decl> ";"
              | <operation_decl> ";"

(13)<attribute_decl> := <attribute_head> <type_spec> <declarator> { "," <declarator> }*

(14)<attribute_head> := "[" ["readonly" ","] "attribute" "]"
                      | "[" "attribute" ["," "readonly"] "]"

(15)<declarator> := <identifier>
                  | <array_declarator>

(16)<array_declarator> := <identifier> <array_size>+

(17)<array_size> := "[" <positive_int> "]"

(18)<positive_int> := <const_expr>

(19)<type_spec> := <simple_type_spec>
                 | constr_type_spec>

(20)<simple_type_spec> := <base_type_spec>
                        | <template_type_spec>
                        | <scoped_name>

(21)<base_type_spec> := <integer_type>
                      | <floating_point_type>
                      | <char_type>
                      | <byte_type>
```

```
                       | <boolean_type>
                       | <string_type>
                       | <any_type>
                       | <type_type>

(22)<template_type> := <sequence_type>
                     | <array_type>

(23)<sequence_type> := "sequence" "<" <type_spec> ">"

(24)<array_type> := <type_spec> <array_size>+

(25)<floating_point_type> := "float"
                           | "double"

(26)<integer_type> := <signed_int>
                    | <unsinged_int>

(27)<signed_int> := "short"
                  | "long"
                  | "hyper"

(28)<unsigned_int> := "unsigned" "short"
                    | "unsigned" "long"
                    | "unsigned" "hyper"

(29)<char_type> := "char"

(30)<type_type> := "type"

(31)<string_type> := "string"

(32)<byte_type> := "byte"

(33)<any_type> := "any"

(34)<boolean_type> := "boolean"

(35)<constr_type_spec> := <struct_type>
                        | <enum_type>
                        | <union_type>

(36)<struct_decl> := "struct" <identifier> [ <struct_inheritance> ] "{" <member>+ "}"

(37)<struct_inheritance> := ":" <scoped_name>

(38)<member> := <type_spec> <declarator> { "," <declarator> }*

(39)enum_decl> := enum <identifier> "{" <enumerator> { "," <enumerator> }* "}"

(40)<enumerator> := <identifier> [ "=" <positive_int> ]

(41)<union_decl> := "union" <identifier> "switch" "(" <switch_type_spec> ")"
                  "{" <switch_body> "}"

(42)<switch_type_spec> := <integer_type>
                        | <enum_type>
                        | <scoped_name>

(43)<switch_body> := <case>+

(44)<case> := <case_label> <element_spec> ";"

(45)<case_label> := "case" <const_expr> ":"
                  | "default" ":";

(46)<element_spec> := <type_spec> <declarator>

(47)<exception_decl> := "exception" <identifier> [ <exception_inheritance> ] "{" <member>* "}"

(48)<exception_inheritance> := ":" <scoped_name>

(49)<module_decl> := "module" <identifier> "{" <definition>+ "}"

(50)<constant_decl> := "const" <const_type> <identifier> "=" <const_expr>

(51)<const_type> := <integer_type>
                  | <char_type>
                  | <boolean_type>
                  | <floating_point_type>
                  | <string_type>
                  | <scoped_name>

(52)<const_expr> := <or_expr>

(53)<or_expr> := <xor_expr>
               | <or_expr> "|" <xor_expr>
```

```
(54)<xor_expr> := <and_expr>
                | <xor_expr> "^" <and_expr>

(55)<and_expr> := <shift_expr>
                | <and_expr> "&" <shift_expr>

(56)<shift_expr> := <add_Expr>
                  | <shift_expr ">>" <add_expr>
                  | <shift_expr "<<" <add_expr>

(57)<add_expr> := <mult_expr>
                | <add_expr> "+" <mult_expr>
                | <add_expr> "-" <mult_expr>

(58)<mult_expr> := <unary_expr>
                 | <mult_expr> "*" <unary_expr>
                 | <mult_expr> "/" <unary_expr>
                 | <mult_expr> "%" <unary_expr>

(59)<unary_expr> := <unary_operator><primary_expr>
                  | <primary_expr>

(60)<unary_operator> := "-" | "+" | "~"

(61)<primary_expr> := <scoped_name>
                    | <literal>
                    | "(" <const_expr> ")"

(62)<literal> := <integer_literal>
               | <string_literal>
               | <character_literal>
               | <floating_point_literal>
               | <boolean_literal>

(63)<boolean_literal> := "TRUE"
                       | "True"
                       | "FALSE"
                       | "False"

(64)<service_decl> := "service" <identifier> "{" <service_member>+ "}"

(65)<singleton_decl> := "singleton" <identifier> "{" "service" <declarator> ";"  "}"

(66)<service_member> := <property_decl> ";"
                      | <support_decl> ";"
                      | <export_decl> ";"
                      | <observe_decl> ";"
                      | <needs_decl> ";"

(67)<property_decl> := <property_head> <type_spec> <declarator> { "," <declarator> }*

(68)<property_head> := "[" {<property_flags> ","}* "property" "]"
                     | "[" "property" {"," <property_flags>}* "]"

(69)<property_flags> := "readonly"
                      | "bound"
                      | "constrained"
                      | "maybeambigious"
                      | "maybedefault"
                      | "maybevoid"
                      | "optional"
                      | "removable"
                      | "transient"

(70)<support_decl> := [ "[" "optional" "]" ] "interface" <declarator> { "," <declarator> }*

(71)<export_decl> := [ "[" "optional" "]" ] "service" <declarator> { "," <declarator> }*

(72)<observe_decl> := "observe" <declarator> { "," <declarator> }*

(73)<needs_decl> := "needs" <declarator> { "," <declarator> }*

(74)<constants_decl> := "constants" <identifier> "{" <constant_decl>+ "}"
```

Glossary

A

Abstraction

The term abstraction denotes the process or the result of a generalization. Generalization describes objects by qualities common to all objects of a certain class of objects. The principle of the generalization is to disregard individual properties of the objects, consequently it is impossible that an abstract object exists anywhere but in theory.

Accessibility

Ability of an application to provide its functionality also in situations where the usage of input and output devices is restricted for some reason. For instance, this can be due to restrictions of the devices, the environment or a physical disability of the user. Often assistive technology is used to provide accessibility, for instance screen readers or braille terminals. From version 1.1, OpenOffice.org has an API for accessibility, which can be used with Java and Gnome accessibility.

Add-In

An add-in is a functional extension for the OpenOffice.org application on the basis of UNO components , which interact with parts of the application that were especially laid out to be extended. Examples of Add-Ins are Chart and Calc Add-Ins.

Any

All purpose data type for variables in UNOIDL. An `any` variable contains whichever data type is specified for UNOIDL.

API

Application Programming Interface. The entirety of published methods, properties and other means for software developers to access an application through software they write using this application.

Assistive Technology

Devices which can be used to improve accessibility, see Accessibility.

AT

Assistive Technology, devices which can be used to improve accessibility. See Accessibility.

Automation

Communication protocol between OLE automation objects. See OLE Automation.

AWT

Abstract Window Toolkit. The OpenOffice.org API contains a module `com.sun.star.awt` with abstract specifications for a window toolkit that handles graphical devices, window environments and user interfaces. In the current implementation of this specification, the specified features are mapped to platform-specific window systems, such as Windows, X Windows or Java AWT. The current C++ implementation is based on the Visual Component Library, a platform independent C++ library for GUIs, which is part of OpenOffice.org.

B

Binary UNO Interface

When method calls are transported over a UNO bridge, a single generic C method is used to dispatch all method calls across the bridge. This method and its parameters is also known as the binary UNO interface.

Bridge

Code that connects different language environments, such as C++, Java and indirectly OpenOffice.org Basic, with each other. The connection is exclusively used to transport method calls with their parameters, and return values back and forth between the language environments.

C

Calendar

Calendaring information in an internationally used application pose the problem to translate between the various calendar systems used in the world. In the context of OpenOffice.org, local calendars are supported through the I18N API.

Calc

OpenOffice.org spreadsheet document or components of the OpenOffice.org application containing the functionality necessary for spreadsheet documents in OpenOffice.org. Although there might be an scalc executable on some platforms, it does not contain the Calc functionality, it starts up a calc document using soffice.exe and its dependables.

Chart

Embedded diagram document or components of the OpenOffice.org application containing the functionality necessary for embedded diagrams in OpenOffice.org. These diagrams visualize numeric and textual data, such as lines, bars, and pies.

CJK

China-Japan-Korea. A group of Asian languages that require similar treatment in user interfaces for common principles, such as the writing direction and other features of Asian document editing.

Class

Class is the description of the common qualities of individual objects in object-oriented languages. This description can be expressed in an object-oriented programming language. A class description may be abstract where it does not contain sufficient implementation to create fully functional instances of a class, or it can be fully implemented. Fully implemented classes are used to create individual instances of objects that act according to the class description.

Client

An object using the services of a server. See server.

Clipboard

The clipboard is common storage place on a computer platform. Information is copied or cut from one application context and transferred to this storage where users paste it into another application context. A variety of file formats can be written to the clipboard making the information useful in many different contexts.

Collation

In the context of OpenOffice.org, ordering of textual information according to ordering rules local to a cultural community. The rules for alphabetical ordering in Latin writing differ from country to country, and there are completely different ordering rules in other cultural communities. OpenOffice.org supports localized collation through its I18N API.

Collection (UNO Collection)

UNO collections are gatherings of objects that are retrieved by enumeration, index or name through collection interfaces. UNO collections are not necessarily UNO containers, because they do not support the addition of new objects to the collection—though a collection can be a container too.

COM

Component Object Model. An object communication framework created as a part of OLE by Microsoft (R) . See OLE.

Command URL

A string containing a command in the OpenOffice.org dispatch framework. See URL.

Commit

Acknowledgment of an open transaction. See transaction.

Complex Text Layout

Complex Text Layout Languages: In CTL languages (such as Thai, Hebrew, Arabic and Indian) multiple characters may combine to form a display cell

Component

The term component has two uses in the UNO context. There are UNO components and XComponents, that is, objects implementing the interface `com.sun.star.lang.XComponent`.

UNO components are shared libraries containing implementations of UNO objects that are registered with and instantiated by a UNO service manager or service factory. If the component only uses a UNO environment, it is a *well formed* component.

An XComponent is a UNO object that allows its owner object to control its lifetime and a user object to register as a listener to be informed when the owner disposes of the XComponent. Occasionally, the term component is used as a shortform for XComponent. For example, since

OpenOffice.org documents loaded by the desktop must always support XComponent, it has become customary to call them components or desktop components. Loaded documents are not UNO components in the sense of a well formed component. They have no shared libraries and cannot be registered and instantiated at a service manager. It should always be clear from the context if the term component means well formed UNO component or XComponent.

Configuration

In the context of OpenOffice.org often used for XML based configuration files. OpenOffice.org has an API to access this configuration, the package installer *pkghck* can insert configuration items, and users can edit the files manually.

Constant

A named value in a computer program that does not change during runtime. Constants are used to handle cryptic parameters in an understandable manner as in `com.sun.star.text.HoriOrientation:LEFT`. Furthermore, if constants are used. it is possible to alter the internal value of a constant without changing every occurrence of this value in written code. But it is not possible to change the value of UNO IDL constants.

Constants Group

A named group of constant values, for example, the group `com.sun.star.text.HoriOrientation` contains constant values that describe possible horizental orientations, such as LEFT, CENTER, and RIGHT. See constant.

Container (UNO Container)

UNO collection of objects with the additional option to add new objects to the collection and to remove objects. See collection.

Connection

An UNO Connection is an open communication channel between a UNO client and server. For example., if a Java program uses OpenOffice.org over the Java language binding, the Java client program connects to the OpenOffice.org application, which then acts as server for the Java client.

A Database Connection is an open communication channel between a database management system and an authenticated user.

Controller

A controller in the frame-controller-model paradigm of OpenOffice.org is a service that controls a visual representation of a OpenOffice.org document model. It may offer interfaces to access the visual representation, but it is not used to change the model it presents. In the frame-controller-model paradigm the view is a hidden implementation detail behind the controller. See frame-controller-model paradigm.

CORBA

Common Object Request Broker Architecture. Platform independent architecture for object communication. CORBA served as one of the examples for UNO.

CTL

see Complex Text Layout

D

DB

Abbreviation for database.

DBMS

Database Management System

DCOM

Distributed Component Object Model. It adds to COM objects the ability to communicate with COM objects on other machines.

DDE

A Windows protocol allowing applications to exchange data automatically. The OpenOffice.org supports DDE through the **Edit – Paste Special** command. OpenOffice.org Basic also uses DDE.

DDL

Data Definition Language. Parts of SQL used to create and alter tables, and modify rules for relational integrity.

Deadlock

A state where two processes wait for another so that they can continue their work. They have to wait until the deadlock is released from outside. For example this can happen if process A locks resource X and process B locks resource Y, and then process B tries to lock resource X and process A tries to lock resource Y.

Desktop

Central management instance for viewable components in the OpenOffice.org application.

Dialog (UNO Dialog)

A UNO dialog shows a window for user input. A dialog contains control elements, such as text fields, buttons, list boxes, and combo boxes. Currently, UNO dialogs are always modal, which means that they must be closed before the process displaying the dialog can continue with its tasks. Furthermore, UNO dialogs do not support data aware controls, rather database connectivity has to be implemented manually. If you want to offer a non-modal window or work with data, consider using a UNO form.

Dispatch Framework

OpenOffice.org has a mechanism that sees documents as targets for uniform command tokens, which are handled for example by documents with methods specific to the document. This alleviates writing a user interface that does not need to know about the internal structure of a document. The user interface asks the document the command tokens it supports, and displays matching menus and toolbars. A toolbar icon like **Background Color** is used for many different objects without knowing in advance about the target object.

The command tokens have to be written in URL notation, therefore they are called *command URLs*, and are sent or *dispatched* to a target frame. The corresponding specification is called *Dispatch API*.

DML

Data Manipulation Language. Part of SQL.

Draw

OpenOffice.org drawing document or components of the OpenOffice.org application which contain the functionality necessary for drawing in OpenOffice.org. Although there might be an sdraw executable on some platforms, it does not contain the actual Draw functionality, it merely starts up a Draw document, using soffice.exe and its dependables.

Draw Page

A layer for graphical objects in OpenOffice.org documents. Each of the document types Writer, Calc, Draw, and Impress have one or multiple draw pages for shapes. Most graphical shapes on a drawpage are geometrical objects, but embedded documents and forms are also located on the draw page of a document.

Document Controller

A part of the frame-controller-model paradigm in OpenOffice.org. The controller of a document is responsible for screen presentation, display control and the current view status of a document.

E

Enum

A named group of predefined values in the OpenOffice.org API comprising all plausible values for a variable in a certain context. Only one enum value can apply at a time. An example for an enum is com.sun.star.text.PageNumberType with the possible values NEXT, PREV and CURRENT.

Enumeration

A collection of UNO objects supporting the interface `com.sun.star.container.XEnumeration` accessed one by one using a loop construction. An XEnumeration has to be created at a `com.sun.star.container.XEnumerationAccess` interface.

Event

In the OpenOffice.org API, an event is an incident between an observable and an observer. The observable sends a message that something has happened that the observer wanted to know about. See listener.

Exception

The exception is a concept for error handling that separates the normal program flow from error conditions. Instead of returning error values as function return codes, an exception interrupts the normal program flow at anytime, transports detailed information about the error and passes it along the chain of callers until it is handled in code. This is helpful for the user to achieve a low-level function, therefore react appropriately, while it is still able to find out exactly what went wrong.

F

FCM

Frame-Controller-Model paradigm. See frame-controller-model.

Filter

There are two kinds of filters in OpenOffice.org, data filters and import/export filters.

Data filters reduce the number of records in a list or database to those records that match the given filter criteria. Examples of filters are those filters in a spreadsheet or database form.

The import and export filters read and write document data for specific file formats. They create OpenOffice.org documents from the files they support in a running OpenOffice.org instance, and create a target file in a supported format from a loaded document.

Form

A form is a OpenOffice.org document with a set of controls that allows users to enter data, and submit the the data to a web server or store them in a database.

Data-aware forms support data-aware controls that display data from a database and write changes to a database automatically. Furthermore, they have built-in filtering and sorting capabilities. It is also possible to create subforms in forms.

Without a connection to a server, forms are only partially useful, because the integration with the surrounding document is still incomplete. Forms cannot be printed well, because text boxes do not shrink or grow, and list boxes and subforms are cut off in printing. It is not possible to have control borders in the user interface and hide them in printing.

Frame

Part of the frame-controller-model paradigm in OpenOffice.org. See frame-controller-model paradigm.

Frame-Controller-Model Paradigm (FCM)

The architectural separation of content, visual representation, and binding to the surrounding window system in OpenOffice.org. Loaded office documents consist of:

- a model object for document data with document manipulation methods
- one or more controllers for screen presentation, display control and current view status of a document model
- one frame per controller that links the controller with the surrounding windowing system, and dispatches command URLs it receives. It makes the document environment exchangeable. For instance, aside from the standard document windows there can be frames for documents in JavaBeans, Browser Plug-ins, MDI Windows, and embedded windows.

Programming with the FCM paradigm is simple: To change the document, use the model. To access the visual representation, ask the controller. To work with the window, obtain the frame.

If you know the Smalltalk model-view-controller paradigm (MVC), it is probably best to see frame-controller-model as a different concept with a few similarities to MVC. The main differences are:

- The controller in FCM incorporates the visual presentation layer: Controller and visual representation are no different objects on API level. It controls the visual representation such as the current page or the visual cursor position, but it is generally not used to control the document content.
- FCM has a frame, which is unknown in MVC.

G

GUI

Graphical User Interface, as opposed to a command line interface. A user interface is the point where a user and a software application meet and interact. A graphical user interface uses graphical representations of commands, status feedbacks and data of an application, and offers methods to interact with it through graphical devices, such as a mouse or tablets.

H

Helpers

Classes or methods with ready-to-use implementations that are used to implement fully functional UNO components. The goal is that implementers of UNO components can concentrate on the functionality they want to deliver, without having to cope with the intricacies of UNO.

I

I18N

Internationalization, written I18N because of the 18 letters between the 'i' and 'n' in internationalization. It provides the functionality to adapt a software to the needs of an international community with their deviating standards. For example, documents should be fully interchangeable, that is, a date should be the same date no matter where the document is edited, but the date needs to be displayed and edited according to the conventions followed in the user's country. Also, the user should be able to combine documents from other countries with his own documents without having to convert date formats.

IDE

Integrated Development Environment is a tool used for software development that integrates editing, debugging, graphical interface design and online help, and advanced features, such as version control, object browsing and project management in a unified user interface. OpenOffice.org contains a small IDE for OpenOffice.org Basic.

IDL

Interface Definition Language is used in environments where interfaces are used for object communication. An interface definition language is frequently used to describe interfaces independently of a particular target language. For instance, CORBA and OLE have their own interface definition languages. UNO does not stand behind these component technologies and specifies its own IDL called UNO IDL.

Implementation

The process of writing a fully functional software according to a specification. Implementation also means the concrete, realized thing as opposed to an abstract concept. For instance, the current version of OpenOffice.org is one possible implementation of the OpenOffice.org API.

Impress

OpenOffice.org presentation document or components of the OpenOffice.org application that contains the functionality necessary for presentation documents in OpenOffice.org. Although there might be an simpress executable on some platforms, it does not contain the Impress functionality, it starts up a presentation document using soffice.exe and its dependencies.

Initialization of UNO Services

UNO objects are initialized when they are instantiated by a service manager if they support the interface `com.sun.star.lang.XInitialization`. The service manager automatically passes the arguments given in `createInstanceWithArguments()` or `createInstanceWithArgumentsAndContext()` to the method `initialize()` of the new UNO object. The service specification for the object documents the arguments if XInitialize is supported.

Instance

An instance is a concrete, individual object specimen created on the basis of an implemented class. In UNO, it is common to ask a service manager for an instance of a service. The service manager chooses a suitable implementation and sets up an object in memory on the basis of this implementation.

Interface

In object-oriented programming environments, the term interface is used for sets of methods that describe aspects of external object behavior in terms of method definitions. The term interface implies that the described aspects abstract from the described functionality. Thus, an interface for a functionality is completely independent of the inner workings of an object that is necessary to support functionality. Interfaces lead to exchangeable implementations, that is, code that is based on stable interfaces works across product versions, while it is relatively easy to extend or replace existing interface implementations.

UNO interfaces have a common base interface `com.sun.star.uno.XInterface` that introduces basic lifetime control by reference counting, and the ability to query an object for an interface it supports.

I/O

Input/Output. The I/O is the physical transfer of byte stream between random access memory and devices that provide data or process data.

J

Java Bean

Reusable software component that can be visually manipulated in builder tools.

Job

UNO component that is set off upon an event. A job component must support the services `com.sun.star.task.Job` and/or `com.sun.star.task.AsyncJob`. Currently there are two ways to activate a job: either by triggering an event at the job executor service or by dispatching a specialized command URL of the protocol *vnd.sun.star.jobs:* at the dispatch framework.

Job Execution Environment

Environment in OpenOffice.org for generic jobs that are implemented as UNO components. A job can be executed upon an event and use configuration data for arbitrary purposes. It is guarded by

the job execution environment which takes care of the job during its lifetime and writes back configuration data after the job has finished its work.

K

L

L10N

Localization, written L10N because of the 10 letters between the 'l' and 'n' in localization. It is the process of adapting a software to the requirements of users in a cultural community or country. For example, this includes translation of user interfaces and the necessary adaptation to the writing used in that community.

Language Binding

Programming language or programming environment that is used with UNO. It is possible to access OpenOffice.org from component technologies, such as OLE, through programming languages.

Listener

Listeners are objects that are set up to receive method calls when predefined events occur at other objects. They follow the observer pattern, that is, an object wants to update itself whenever it observes a change in another object registers with the object it wants to observe, and is called back when the prearranged event occurs at the observed object. The observable maintains a list of observers that want to be notified about certain events. This pattern avoids constant polling and ensures that observers are always up-to-date. Listeners in OpenOffice.org are specialized for the UNO environment. A listener implements a UNO listener interface with predefined call back methods. This interface is passed to the corresponding event broadcaster in an addXXXListener() method. The broadcaster calls methods on this interface on listener-specific events. The callback methods of a listener take an object that is derived from the base event struct `com.sun.star.lang.EventObject`. This object contains additional information about the event that lead to the listener callback.

Locale

A locale is a string which uniquely identifies a specific cultural community, defined by the *country* where a community lives, and by the *language* spoken. In the I18N API of OpenOffice.org, a locale consists of two parts encoded as `<language>_>COUNTRY>`: a two-letter language code (ISO-639) and a two-letter country code (ISO-3166). Examples are `en_US` for American English with American date, time, measuring and currency conventions, `en_UK` for British English and British conventions, `de_DE` for German as spoken in Germany with German conventions, `es_ES` for Spanish as spoken in Spain, `es_MX` for Spanish as spoken in Mexico. Locales sometimes occur with a third *variant* part which is used to denote further sub-divisions and variants, such as `es_ES_TRADICIONAL` for Spanish with traditional collation rules, as opposed to modern collation. The variant part is user-dependent.

M

Math

Math is the embedded formula document or components of the OpenOffice.org application that contains the functionality necessary for embedded formulas in OpenOffice.org. Formula documents create mathematical formulas based on a meta description.

Model

The Model is an object representing document data and document manipulation methods, and is part of the frame-controller-model paradigm. See frame-controller-model paradigm.

Module

In UNO IDL, a module is a namespace for type definitions. The OpenOffice.org API is divided in 55 modules, such as awt, uno, lang, util, lang, text, sheet, drawing, presentation, chart, and sdb. The modules text, sheet drawing and presentation do not map directly to Writer, Calc, Draw and Impress documents, but the interfaces in these modules are used across all document types.

MVC

The Model-View-Controller paradigm that is the separation of document data, presentation and user interaction into independent functional areas. The frame-controller-model paradigm in OpenOffice.org has been designed with MVC in mind.

N

O

Object

As a general term, an object in the context of this manual is an implemented class that is instantiated and has methods you can call. A UNO object is an object with the ability to be instantiated in the UNO context and to communicate with other UNO objects. For this purpose, it supports the UNO base interface `com.sun.star.uno.XInterface` in addition to the interfaces for the individual functionality it offers.

Object Identity

In UNO, a comparison of object references must be true for all references to an identical object. This rule is called object identity.

OLE

Object Linking and Embedding. It is a set of various technologies offering an infrastructure for object communication across language environments, and is indigenous on the Windows platform. In Inside OLE (Redmond 1995), Kraig Brockschmidt defines OLE "OLE is a unified environment of object-based services with the capability of both customizing those services and arbitrarily extending the architecture through custom services, with the overall purpose of enabling rich integration between components."

Among others, OLE comprises compound documents, visual editing, OLE Automation, the Component Object Model and OLE controls. Moreover, the term OLE as a collective term for a number of technologies has been superseded by ActiveX, which comprises even more technologies.

Although there are implementations for certain aspects of OLE on other platforms, Windows is the primary OLE platform. OpenOffice.org supports a certain aspect of OLE Automation,that is, OpenOffice.org is an OLE Automation server that offers the complete OpenOffice.org API to Automation clients.

The term OLE is sometimes used for document embedding techniques within OpenOffice.org. OpenOffice.org documents are embedded into each other, and appear as "OLE Objects" on draw pages. That means, they are edited in place, and act like embedded OLE documents, but the platform infrastructure for OLE is not used. Therefore, this also works on platforms other than Windows. Real OLE objects are handled differently, the embedded object is handed to the application which is registered for the embedded document and opened in an independent application window.

OLE Automation

Automation is the part of the OLE technology that allows developers to call methods in applications supporting OLE automation. An OLE application publishes methods to be used from other OLE enabled applications. The called application acts as server, and the caller as client in this relationship. Under Windows, a OpenOffice.org application object is available that offers almost the complete OpenOffice.org API to automation clients.

P

Package Installer

In order to facilitate the modular extension of OpenOffice.org by custom components, OpenOffice.org offers a commandline tool named *pkgchk* which can be used to deploy component files, insert Basic libraries and alter the configuration of an existingOpenOffice.org installation.

pkgchk

See package installer.

Prepared Statement

Precompiled SQL statement that are parameterized and sent to a DBMS.

Protocol Handler

UNO component that handles custom URL protocols. A URL protocol is the part of a URL that stands before the colon, as in *ftp:* (file transfer protocol) or *http:* (hypertext transfer protocol). This mechanism is used as of OpenOffice.org version 1.1 to integrate OpenOffice.org extensions into the user interface. For example, a menu item can be configured to dispatch a command URL *vnd.company.oo.newcomponent:NewFunction*. A protocol handler for the protocol *vnd.company.oo.newcomponent:* could route this command to the corresponding routine `newFunction()`. This technique also forms the basis for the job execution environment, where *vnd.sun.star.jobs:* URLs are routed to components that support suitable job interfaces.

Q

Query

See database query, query interfaces, query adapter.

R

Redline

Text portion in a text document that reflects changes to a text document.

Reference Counting, Ref Counting

Controlling the lifetime of an object by counting the number of external references to the object. A ref counted object is destroyed automatically when the number of external references drops to zero.

Registry Database

Backend repository that contains information about UNO components registered with the service manager.

Rollback

Is the rejection of an open transaction. The data are restored to the state before the transaction was started. See transaction.

Ruby

Asian text layout feature, similar to superscript and subscript in western text. See *www.w3.org/TR/ruby/*.

S

SAL

System Abstraction Layer. C++ wrappers to system-dependent functionality. UNO objects written in C++ use the types and methods of SAL to create platform-independent code.

Sequence

Sequence is a set of UNO data types accessed directly without using any interface calls. The sequence maps to arrays in most language bindings.

Server

A server is an object that offers services to clients. OpenOffice.org frequently acts as server when it is accessed through UNO, but it can also be a client to UNO components, instantiating and using UNO objects in another application. The simplest use for OpenOffice.org calling objects in other processes are listener callbacks. See client.

Service (UNO Service)

A UNO service describes a UNO object by combining *interfaces* and *properties* into an abstract object specification. This definition of the term service is specific to UNO, therefore do not confuse it with the general meaning of the word service in "a server offers services to its clients".

Service Manager

Factory for UNO services. A service manager supports the service `com.sun.star.lang.ServiceManager`, and its main task is to provide instances of UNO objects by their service name. This is done by factory methods that take a service name and optional arguments. The service manager looks in its registry database for UNO components that implement the requested service, chooses an implementation and uses a component loader to instantiate the implementation. It finally returns the interface `com.sun.star.uno.XInterface` of the new instance.

Singleton

Singletons specify named objects. Only *one* instance exists during the lifetime of a UNO component context. A singleton references one service and specifies that the only existing instance of this service is reached over the component context using the name of the singleton. If no instance of the service exists, the component context instantiates a new one.

Specification

Is an abstract description of qualities required for a certain task. The realization of a specification is its implementation.

SQL

Structured Query Language, pronounce SEE-KWEL. A standard language for defining databases, and for editing data in a database. SQL is used with relational database management systems.

Statement

An object in the sdbc module of the OpenOffice.org API that encapsulates a static SQL statements. See prepared statement.

Stored Procedure

The server-side process on a SQL server that executes several SQL commands in a single step, and is embedded in a server language for stored procedures with enhanced control capabilities.

Style

A predefined package of format settings applied to objects in OpenOffice.org documents.

Subform

Database form that depends on a main form. Usually a subform is used to display selected data, matching to the current record of the subform, for example, a main form could show a company address, and a subform could list the contact persons in that company. When a user browses through the companies in the main form, the subform is constantly updated to show only the contacts in the current company. This is achieved by a parameterized query in the subform, which takes a unique key from the main form and selects multiple records that match this key.

SVG

Scalable Vector Format. A W3C specification for a language describing two-dimensional vector, and mixed vector or raster graphics in XML. See *www.w3.org/TR/SVG/*.

T

Thread

Computer programs in single-task operating systems have a predefined course with a defined starting and ending point. Between these points, it is clear which instruction the CPU is currently executing, and that the next instruction in the program will be executed next by the CPU. On pre-emptive multi-tasking systems, the ability of modern CPUs to switch their current execution context is used to spawn sub-processes that run simultaneously with the original process. These sub-processes are called threads. In this situation, the CPU always knows which instruction it executes next, but the applications do not know if the CPU will execute their next instruction after the current instruction. Other threads might alter commonly used data. This makes it necessary to write thread-safe programs. A thread-safe program is aware that other threads might interfere with the current thread, and take precautions to shield commonly used data from other threads.

Transaction

A batch of SQL commands that are considered a unity. All commands must be executed successfully, or the data must be restored to the state before the transaction was started. When using transactions, you tell the DBMS that it should start a transaction, then issue all SQL commands you need. After all the commands have been executed, commit the transaction. If an error occurred during one of the commands, restore the previous state by telling the DBMS to roll back the transaction. Transactions can become tricky, because your process or other processes can have open transactions in which they are altering data and locking rows. Therefore, plan carefully if you want to see changes before they are committed, or ensure that the data does not change when you read them again (transaction isolation).

Transliteration

Conversion of characters according to conversion rules that are valid for a cultural community, such as case conversions, conversions between Hiragana and Katagana, and Half-width and Full-width.

Type Mapping

The UNO interface definition language uses meta types for its type definitionsare mapped to types of a real programming language. How the UNO IDL types are mapped is defined by the language binding for a target language.

U

UCP

Universal Content Provider. Subystem of the UCB for one particular storage system or data source.

UCB

Universal Content Broker. Unification layer for access to storage systems or data sources, such as file, ftp, and webDAV.

UI

User Interface. See GUI.

Unicode

Unicode is a standardization effort by the Unicode consortium to provide a unique number for every character, regardless of platform, program and language. See www.unicode.org.

UNO IDL

UNO Interface Definition Language. See IDL.

UNO

Universal Network Objects. Platform-independent component technology used as a basis for OpenOffice.org.

UNO Component

See component.

UNO Collection

See collection.

UNO Container

See container.

UNO Dialog

See dialog.

UNO Object

See object.

UNO package

Packaged zip archive containing files for the package installer.

UNO Proxy

A UNO proxy (proxy is used as a shortform) is created by a bridge and is a language object that represents a UNO object in the target language. It provides the same functionality as the original UNO object. There are two terms which further specialize a UNO proxy. The UNO interface proxy is a UNO proxy that represents exactly one interface of a UNO object, whereas a UNO object proxy represents an UNO object with all of its interfaces.

URL

Uniform Resource Locator. In addition to the public URL schemes defined in RFC 1738, OpenOffice.org uses several URL schemes of its own, such as command URLs for the dispatch API, UNO Connection URLs for the `com.sun.star.bridge.UnoUrlResolver` service, private:factory URLs for the interface `com.sun.star.frame.XComponentLoader` and database URLs to create database connections, `com.sun.star.sdbc.XDriverManager`.

V

VCL

Visual Component Library. Platform-independent C++ library that handles GUI elements. Part of OpenOffice.org.

View

A view is the presentation of document data in a GUI. In the OpenOffice.org frame-controller-model paradigm, there are no view objects separate from controllers, but the controller contains the view it controls.

W

Weak Reference

Reference to a UNO object which has to be converted to a hard reference before it can be used. A weak reference automatically turns into a null reference when the referred object is destroyed, and it does not keep the referred object alive.

Writer

The Writer is the OpenOffice.org word processor document or components of the OpenOffice.org application containing the functionality necessary for word processing in OpenOffice.org. Although there might be an swriter executable on some platforms, it does not contain the actual Writer functionality, it starts up a Writer document using soffice.exe and its dependables.

X

X<Interface Identifier>

Prefix for UNO Interfaces.

XML

Extensible Markup Language. Multitude of standards developed by the W3C for the definition and the processing of structured file formats. See *www.w3.org/XML/*

Y

Z

Index

__getServiceFactory() 202
__writeRegistryServiceInfo() 202, 206
_blank 341, 358
_default 358
_parent 341, 358
_self 341, 358
_top 341, 358
- 189
/ 189
// 190
/// 190
/* 190
/** 190
.dll 279
.idl 192
.pba 787
.rdb 279
.so 279
.uno 276
.urd 192
.xcu 280
.xlb 279, 789
.xlc 789
^ 189
~ 189
$(home) 444
$(inst) 444
$(instpath) 444
$(insturl) 444
$(lang) 444
$(langid) 444
$(path) 444
$(prog) 444
$(progpath) 444
$(progurl) 444
$(temp) 444
$(user) 444
$(userpath) 444
$(userurl) 444
$(vlang) 444
$(work) 444
* 189
& 189
% 189
+ 189
<< 189
>> 189
| 189

A

absolute() 839
Abstract Window Toolkit (AWT) 327
Acceptor 78
acceptsURL() 872
accessibility 959
accessibility objects 960
accessibility tree 960
acquire() 197, 224
actions 976
active document model 334
active frame model 334
ActiveX 146
4.7.3 Add-Ons 256
  configuration 258
  installation 269
AdministrationProvider 926, 928f.
afterLast() 839
aggregation 201
AnimationEffect 693
any 46, 110, 182
API reference 74
API Reference 30
appendFilterByColumn() 805
appendOrderByColumn() 805
applicat.rdb 206, 214

application environment 325
archive files 193
array 191
assistive technology 959
asynchronous call 77
AT (assisitve technology) 959
attribute [instruction] 182
AutoCorrect 435
autodoc 178
Automation
accessing properties 150
bridge services 172
calling functions 150
client-side conversions 160
conversion mappings 159
DCOM 170
default mappings 157
errorcodes 164
exceptions 164
interfaces 154
mapping of any 161
mapping of sequence 162
mapping of string 162
registry entries 149
service manager component 148
Service Manager Component 148
Simple Types 156
structs 154
type mappings 156
usage of types 154
value objects 162
Windows Script Components 170
Windows Scripting Host 170
WSC 170
WSH 170
automation bridge 145
AutoPilot 436, 734
AutoText 435

B

backup copies 435
11.3 Basic 128, 296, 757
accessing the UNO API 760
accessing UNO services 129
adding event handlers 732
AutoPilot dialogs 734
11.2.2 Basic IDE window 743
Basic editor mode 743
dialog editor mode 743
calling a sub 729
case sensitivity 141
constant groups 141
creating a module 726
creating dialogs 730
creating dialogs at runtime 781
date functions 758
debugging a Basic UNO program 728
design tools window 731
enums 141
exception handling 142
file I/O 757
information about UNO objects 131
instantiating UNO services 130
11.4 library organization 764
accessing libraries 765
Creating a Link to an Existing Library 768
creating a new library 768
library 764
library container 764
library container API 767
library container properties 765
library elements 764
loading libraries 766
variable scopes 769
listeners 143
numeric functions 759
runtime library functions 757
screen I/O functions 757
sequences and arrays 137
simple types 135
source editor window 727
special behavior 762
special behaviour
rescheduling 762
threads 762
string functions 759
structs 140
time functions 758
writing a Basic UNO program 728
Basic dialogs 725
11.2 Basic IDE 735
dialog editor 748
macro dialog 736
macro organizer dialog 738
libraries tab page 740
managing Basic and dialog libraries 736
modules tab page 738
Basic libraries 279
Basic library container index file 787
Basic library index file 787

Basic macros 725
beforeFirst() 839
bitmap 976
bookmarks 436
boolean 45, 182
Bootstrap 75
bootstrapping 286
bound [property flag] 186
Braille terminal 959
breakpoint 728
BridgeFactory 78
bridges 63, 960
broadcaster 961
byte 45, 182

C

C++ 145, 296
C++
establishing interprocess connections 121
exception handling 127
file access 119
mapping of any 122
mapping of sequence 124
mapping of type 126
Simple Types 122
system abstraction layer 119
thread synchronization 120
threads 120
threadsafe refcounting 119
type mappings 122
weak references 126
C++ Binding 117
Calc 793
cancelRowUpdates() 841
caret 970
case conversion 384
ccessibility API 959
cell
accessing 564
cell range
accessing 564
array formulas 570
data array 566
merging 565
multiple operations 569
operations 568
properties 564
chain of responsibility. 368
changesOccurred() 938
char 44, 182
Chart3DBarProperties 718
ChartData 709f.
ChartDataArray 709f.
ChartDocument 709
charts
3-dimensional 718
Add-Ins 720
apply an Add-In 722
axis 714
chart type 708
creating charts 705
data access 710
data point 716
data series 716
default type 708
Diagram 713
document controller 720
document model 709
legend 709
pie charts 719
statistical properties 717
stock charts 719
titles 709
working with charts 709
class files 192
clearParameters() 847
6.2.1 clipboard 378
becoming a clipboard viewer 382
copying data 380
data formats 383
pasting data 379
coding styles 321
colors (user-defined) 436
Column 807, 858
columns 807
ColumnSettings 810
Command 906
command execution 367
command tokens 367
command URL 331, 367
CommandType 847
comments 190
communication process 367
compatibly 277
compiler 180
component 70, 338
component context 84

component framework 327, 337
component operations 193
component window 328
component_getFactory() 226
component_writeInfo() 227
4 components 63, 177
  add 278
  architecture 193
  debugging 215
  deployment options 278
  installing manually 285
  registration 214
  Registration 282
  troubleshooting 217
configmgr.ini 929
configmgr.rc 929
configuration data files 280
configuration files 436
configuration layers 924
configuration management 923
configuration schema files 280
ConfigurationAccess 926f., 932
ConfigurationProvider 926, 928
ConfigurationUpdateAccess 926, 928, 935
Connection 803, 811, 872
connection pooling 820
ConnectionPool 820
connections 811
Connector 78
connectWithCompletion() 814
const [UNOIDL] 189
constant groups 141
constant groups) 45
constrained [property flag] 186
container
  enumeration container 50
container window 328
containers 95
controller 330
controller object 338
Controllers 344
ControlShape 885
Corba 319
CORBA 106
CORBA IDL 63
core interfaces 194
cppumaker 178, 192, 284
createInstance() 199
createInstanceWithArguments() 200, 209, 228
createInstanceWithArgumentsAndContext() 200, 209, 228
createRegistryServiceFactory() 290
createStatement() 829
cursor 970
CustomPresentationAccess 690
cyclic references 104

D

15.3 data source 928
  connecting to 928
  using 930
Data Source Administration [dialog] 796
DataAwareControlModel 894
database 793
database design 848
Database Management System (DBMS) 829
DatabaseContext 796
DataDefinition 813
DataSource 798
DBMS features 868
DCOM 170
dcomcnfg.exe 170
DDL 850
debugging 215
defaultBootstrap_InitialComponentContext() 286
DefaultControl 886
defining
  service 185
DefinitionContainer 800
DeleteRows 833
deployment options 278
descriptor pattern 865
design mode 881
design patterns 321
Desktop 333, 901
desktop environment 325
desktop frame 326
desktop object 326, 336
Diagram 708
11.5.2 dialog controls 773
  check box 774
  combo box 776
  command button 773
  currency field 780
  date field 779
  file control 781

formatted field 780
group box 778
image control 773
label field 774
line 779
list box 776
numeric field 780
option button 774
pattern field 780
progress bar 778
scroll bar 777
text field 775
time field 779
11.5.1 dialog handling 770
dialog as control container 771
getting the dialog model 771
showing a dialog 770
dialog library container index file 787
dialog library index file 787
dialog properties 772
dialog-lb.xml 787
dialog-lc.xml 787
dictionaries 436
dictionaries (custom) 436
Dim3DDiagram 718
4.7.4 disable commands 270
at runtime 273
configuration 272
disabled 959
dispatch communication 374
6.1.6 dispatch framework 230, 330f., 367
dispatch process 369
processing chain 368
status information 369
dispatch framework 327
dispatch interception 373
dispatch process
dispatch results 373
dispatching a command 372
getting a dispatch object 370
listening for context changes 372
dispose() 200
DisposedException 33, 75, 107, 121
Documents
closing 360
loading 352
target frame 358
URL Parameter 357
loading [example] 359
printing 367
storing 365
double 45, 182
double-checked locking 321
Draw 653
9 drawing document 653
creating 657
exporting 659
loading 657
page handling 664
page partitioning 665
printing 661
shapes 665
storing 658
DrawingDocumentDrawView 702f.
DrawPage 664, 688, 692
driver
Adabas 816
ADO 816
dBase 816
Flat file format (csv) 816
JDBC 816
Mozilla addressbook 817
ODBC 3.5 816
Driver 816, 871
DriverManager 814
dynamic link libraries 193

E

enum 190
enumeration types 45
enums 116, 141
error 188
event 826
event listeners 97
event names 430
EventObject 826
events 97, 400
OnCloseApp 435
OnFocus 431
OnLoad 430
OnModifyChange 431
OnNew 430
OnPrepareUnload 431
OnPrint 431
OnSave 430
OnSaveAs 430
OnSaveAsDone 430
OnSaveDone 431

OnStartApp 435
OnUnfocus 431
OnUnload 431
exception 73, 431
Exception 98, 115, 127, 142, 164
exception [UNOIDL] 188
exception handling 98, 142
exceptions 115, 164
executeUpdate() 831, 852
exit 334
export filter 404, 419
extended type detection 409
external icons 436

F

FadeEffect 692
FillProperties 678
filter 410
configuring 413
deep detection 425
export 404
filter section 423
flat detection 425
import 404
loading 406
media descriptor 408
options 412
PocketWord 423
properties 416
storing to a URL 407
type section 423
XML filter detection 424
filter development 419
filters 436
first() 839
float 45, 182
form
data awareness 890
filtering and sorting 893
sub forms 891
Form 884, 890
Form Components 888
form document
focussing controls 882
locating controls 882
FormComponent 888
FormComponents 883
FormControlModel 885, 888
Forms 879
frame loader 328, 410
number formats 427
properties 419
frame object 326
Frame-Controller-Model (FCM) 327
Frames 339
actions 341
active frame 342
assigning windows 350
creating 350
creating [example] 351
current component 342
custom name 340
Frame Hierarchies 340, 351
Frame setup 340
frames supplier 351
status indicator 343
sub-frames 343
top-level frame 341
framework API 325

G

Gallery database 436
generic communication 367
GenericDrawPage 701
getCatalogs() 849
getCatalogTerm() 848
getColumns() 809, 849
getComposedQuery() 805
getConnection() 814
getConnectionWithInfo() 815
getDatabaseProductVersion() 848
getDate() 836
getDriverMajorVersion() 848
getDriverMinorVersion() 848
getFilter() 805
getIdentifierQuoteString() 850, 852
getImplementationId() 225
getImplementationName() 198
getMaxCharLiteralLength() 849
getMaxColumnsInTable() 849
getMaxConnections() 849
getMaxRowSize() 849
getMaxStatementLength() 849
getMaxTablesInSelect() 849
getMetaData() 872
getNumericFunctions() 875

getOrder() 805
getPrimaryKeys() 849
getProcedureColumns() 850
getProcedures() 850
getProcedureTerm() 848
getQuery() 805
getRow() 839
getSchemas() 849
getSchemaTerm() 848
getServiceFactory() 205
getSQLKeywords() 849
getString() 836
getStringFunctions() 875
getStructuredFilter() 805
getSupportedServiceNames() 199
getTables() 808, 849
getTypes() 198, 225
getUDTs() 850
getURL() 848
getUserName() 848
Gnome access bridge 960
Gnome Accessibility API 959f.
GNU make [command] 178
GraphicExportFilter 659
Gregorian calendar 384
Group 863
GroupDescriptor 868
GroupShape 676
GSS-API 452
GUI event 332

H

header files 192
help files 436
Hindi 386
Hiragana 385
home directory 442
HTMLForm 890
hyper 45, 182
hyperlink 977
hypertext document 977
hyphenator 397

I

idlc 178, 192, 284
idlcpp 178
image 976
implementation name 198
import filter 404, 419
Impress 653
incompatibly 277
index 867
index entries 386
index service 859
IndexColumn 858
indirectly compatibly 277
initialize() 210
intercepting context menus 375
interface 38, 112
  core interfaces 194
  defining 182
  implementing own interfaces 204
interface [instruction] 185
interfaces 65
internationalization 383
interprocess connection 121
isAfterLast() 839
isBeforeFirst() 839
isFirst() 839
isLast() 839
ISO-3166 387
ISO-639 387

J

jar files 177
Java 296
Java
  language binding 107
  mapping of any 110
  mapping of constants 116
  mapping of enums 116
  mapping of exceptions 115
  mapping of interface 112
  mapping of method parameters 112
  mapping of module 112
  mapping of sequence 112
  mapping of structs 114
  service manager 107
  type mappings 109
Java Accessibility API, 959
Java archive files 279
javamaker 178, 192, 284
job execution environment 230

4.7.2 jobs 243
arguments 248
asynchronous 247
configuration data 251
environment 248
execution environment 245
implementation 246
initialization 248
installation 253
lifetime control 246
returning results 250
supported events 255
synchronous 247
wrapper object 246
JScript 145

K

Kerberos 452
Key 861
key strokes 977
keyboard 959
KeyColumn 858
KeyRule 861
KeyType 861

L

language bindings 106
last() 839
LDAP 793
libraries
application libraries 785
application library container 785
storage 784
with password protection 786
without password protection 786
library deployment 789
LineProperties 678
linguistic API 394
listener 143, 961
listener interfaces 97
listening mode 75
live mode 881
locale dependent data 383
locking (double-checked) 321
long 45, 182

M

make [command] 178
MAPI 793
maybeambiguous [property flag] 186
maybedefault [property flag] 186
maybevoid [property flag] 186
media descriptor 421
method parameters 112
MIDL 63
mode
design mode 881
live mode 881
model object 338
model-view paradigm 879
Model-View-Controller (MVC) 327, 770
Models 346
active controller 347
modified status 347
module 112
module: [instruction] 181
modules 72
monitor 959
mouse 959
moveToCurrentRow() 840
moveToInsertRow() 840
multi paths 435
multi-threaded 321
multimedia files 436
mutex 321

N

next() 835, 838
nullsAreSortedHigh() 848
nullsAreSortedLow() 848
number formats
applying 428
managing 427

O

object identity 106
office component 338
office component 330
office components 326
OLE 146
OLE Automation Bridge 297

OLE2Shape 709
OleApplicationRegistration 174
OleBridgeSupplier2 172
OleBridgeSupplierVar1 174
OleObjectFactory 174
oneway call 77
onstants 116
optional [property flag] 186

P

password cache 455
path
  list of paths 443
  part of a path 443
  single path 443
path settings service 435
path substitution service 442
path variables 442
patibly 277
patterns (user-defined) 436
pipe 421
pkchkg 243
pkgchk 178, 276, 278, 789
Plugins 436
PolyPolygonBezierDescriptor 671
predefined queries 802
prepareCommand() 847
prepareStatement() 846
preprocessing 180
presentation document
  animations and interactions 693
  custom slide show 690
  graphics styles 697
  loading 686
  page formatting 701
  presentation effects 692
  presentation styles 699
  printing 687
  settings 689, 700
  slide transition 692
  zooming 703
previous() 838
printer 959
PrinterDescriptor 661
printing
  page breaks 546
  print areas 546
  print settings 545
  scaling 546
PrintOptions 661
property 38
property [instruction] 185
PropertySet 700
4.7.1 protocol handler 230, 232
  C++ 236
  configuration 241
  installation 243
  Java 235
Python components 279
PyUNO 297, 314

Q

query 800
Query 803
QueryComposer 803
QueryDefinition 800
queryInterface() 197

R

rdbmaker 178, 284
readonly [property flag] 186
refreshRow() 844
regcomp 178, 227, 237, 283
regcomp (tool) 316
regcompare 284
regfilter.bas 414
regfilter.ini 414
registration 214
registry 149
registry database 192, 206, 214, 217, 227
regmerge 178, 192, 283f.
regview 178
relative() 839
release() 197, 224
remote calls 77
removable [property flag] 186
request 368
ResultColumn 834
ResultSet 828, 832
ResultSet cursor 835
ResultSetMetaData 845
return values 152
rfc1510 452

rfc2743 452
RotationDescriptor 674
RowSet 794, 822, 891
run() 201
Runtime Environment 106
RuntimeException 98, 115, 127, 142, 164

S

SAX 421
scalar functions 875
screen magnifier 959
screen reader 959
script type 384
script-lb.xml 787
script-lc.xml 787
sdb module 803
SDBC 793
SDBC driver 870
SDBCX 853
selection 969, 974
sequence 48, 72, 112
sequence [UNOIDL] 187
service [instruction] 185
service implementations 70
4.5.6 service manager 84, 148, 209
  bootstrapping 286
  dynamically modifying 288
  special configurations 288
Service Manager 34
service manager component 148
Servicemanager 33, 75, 121, 130
ServiceManager 84, 107, 148
services 38, 67
setFilter() 805
setOrder() 805
setQuery() 805
Shape 669, 693
ShapeCollection 676
shapes
  Bezier shapes 671
  binding 676
  combining 676
  connectors 684
  drawing properties 678
  glue points 684
  grouping 676
  inserting files 686
  layer handling 685
  moving 674
  navigating 686
  ordering 676
  rectangle shape 667
  rotating 674
  scaling 674
  shadow 684
  shape types 668
  shearing 674
  transforming 675
shared libraries 177, 276, 279
short 45, 182
shutdown process 334
simple screen reader 963
Single Factory 205
single path 435
Single Sign-On API 451
singleton [instruction] 191
singletons 73
soffice 75, 107
Software Development Kit (SDK) 178
spellcheck 436
spellchecker 396
spreadsheet
  add-ins 648
spreadsheet add-ins 435
8 spreadsheet document 537
  cell
    annotations 573
    errors 571
    formulas 571
    properties 571
    styles 635
    text content 572
  cell range 553, 564
  cells 571
  columns 561
  copying cell ranges 563
  creating 540
  document model 537
  drawpage 539
  filter options 542
  inserting cells 563
  loading 540
  moving cell ranges 563
  naming 563
  page breaks 563
  printing 545
  properties 562

rows 561
saving 541
service manager 538
services 551
sheet cell 557
spreadsheets container 538
8.4.1 styles 634
page 636
SQL 793
SQL statement 829
SQLQueryComposer 805
SSO (Single Sign-On) 451
SSO password cache 455
SSR 963
Star Database (SDB) 794
Star Database Connectivity (SDBC) 793f.
Star Database Connectivity Extension (SDBCX) 794
StarOffice 5.x 326, 336
Statement 829
states 975
StockDiagram 719
storesMixedCaseQuotedIdentifiers() 852
string 45, 182
struct 71
structs 114, 140
supportsAlterTableWithDropColumn() 849
supportsANSI92EntryLevelSQL() 849
supportsBatchUpdates() 849
supportsCoreSQLGrammar() 849
supportsFullOuterJoins() 849
supportsMixedCaseQuotedIdentifiers() 849
supportsPositionedDelete() 849
supportsService() 199
supportsStoredProcedures() 849
supportsTableCorrelationNames() 849
synchronous call 77
system abstraction layer 119
system pointer 331

T

Table 807
tables 807
tables
database tables 537
spreadsheets 537
text tables 537
temp-files 436
templates 436
terminate listener 334
terminate office 334
7 text document 457
auto text 473
block user interaction 533
bookmarks 503
chained text frames 515
character properties 475
columns 531
control characters 470
controller 533
cursor properties 481
document model 457
embedded objects 516
endnotes 509
footnotes 509
graphic objects 516
hyperlink properties 480
index marks 507
indexes 504
inserting text files 473
line numbering 529
link targets 533
loading 463
model cursor 461
number format 529
outline numbering 526
paragraph numbering 526
paragraph properties 475
printing 465
redline 519
reference marks 508
ruby text 520
saving 464
search and replace 482
shape objects 511
sorting 473
7.4.1 styles 520
character styles 522
frame styles 523
numbering styles 523
page styles 523
paragraph styles 523
text field 497
text frame 514
text section 529
view cursor 461
visible cursor 461, 535
visible cursor position [code sample] 461
Text Indices 970

text tables 485
accessing existing tables 497
autoformatting 491
charting 491
inserting 492
naming 491
properties 491
sorting 491
text type 969
TextProperties 678
Thai 386
thesaurus 399, 403
thread 321
thread identity 77
thread synchronization 120
threads 120
toolbar 436
toolkit 331
TransactionIsolation 868
transient [property flag] 186
trivial component 330
trivial components 326
type detection 409
extended type detection 409
type library files 279
TypeClass 64
TypeDetection.xcu 420f.
TypeInfo 207

U

UAA 959
UCB 901
14.4 UCB API 904
accessing content 905
content commands 906
content properties 907
content provider proxies 920
copying contents 916
creating contents 913
deleting contents 915
documents 911
folders 909
instantiating the UCB 905
linking contents 916
moving contents 916
preconfigured UCBs 919
UCP registration information 917
unconfigured UCBs 917
UCP 901
unicode type 384
Uniform Resource Identifier 901
union 191
Universal Content Broker 901
Universal Content Provider 901
UniversalContentBroker 901, 905
uno 178, 290
3 UNO 63
Basic 128, 296
binary UNO 299
bootstrapping 300
Bridge 299
bridging language 298
C++ 117, 296
coding styles 321
collections 95
component context 84
component loader 300
components 177
containers 95
design patterns 321
event listeners 97
event model 97
events 97
3.3.7 exception handling 98
runtime exceptions 99
user-defined exceptions 98
interface bridge 299
interface proxy 299
interprocess connection
asynchronous call 77
closing a connection 81
connection aware client [example] 82
creating the bridge 80
importing an object 76
interprocess bridge 77
listening mode 75
oneway call 77
opening a connection 78
synchronous call 77
thread identity 77
UNO URL 76
interprocess connections 75
Java 296
Java language binding 107
language bindings 106, 298
language object 298
lifetime of UNO objects 100
listener interfaces 97
object bridge 299

object identity 106
object proxy 299
propagation of component contexts 87
proxy 298
Reflection API 306
runtime environment 106
service manager 84
target environment 298
target language 298
using UNO interfaces 89
weak objects 104
weak references 104
UNO access bridge 960
UNO Accessibility API 959
UNO Executable 290
UNO IDL 63
UNO package structure 279
UNO Runtime Environment 106
UNO URL 76
UnoControl 880
UnoControlModel 880, 888
4.2 UNOIDL 179
array 191
attributes 182
comments 190
const 71, 189
constants 72, 189
enum 72, 190
error 188
exception 73, 188
generating source code 192
inheritance 188
interfaces 65
modules 72
operations 184
preprocessing 180
sequence 72, 187
service
defining 185
services 67
including properties 69
referencing interfaces 68
referencing other services 70
simple types 64
singleton [instruction] 191
singletons 73
struct 71, 187
type any 65
union 191
UNOIDL compiler 180
UnoUrlResolver 33, 75, 107, 121
unsigned hyper 45, 182
unsigned long 45, 182
unsigned short 45, 182
updateFloat() 841
updateRow() 841
URE (UNO runtime environment) 106
URI 901
User 865, 867
user settings 436
user-defined colors 436
user-defined patterns 436
UserDescriptor 867
usesLocalFilePerTable() 848
usesLocalFiles() 848

V

VB Script 145
View 863
Visual Basic 145
void 182

W

weak objects 104
weak references 104
window 339
Window
interfaces 349
window handle 375
window peer 339
Windows Script Components 170
Windows/Java access bridge 960
work folder 436
working directory 442

X

XAcceptor 78
XAccessibleAction 976
XAccessibleComponent 968
XAccessibleContext 966
XAccessibleEditableText 970
XAccessibleEventBroadcaster 972
XAccessibleEventListener 973
XAccessibleExtendedComponent 969

XAccessibleHyperlink 977
XAccessibleHypertext 977
XAccessibleImage 976
XAccessibleKeyBinding 977
XAccessibleRelationSet 974
XAccessibleSelection 974
XAccessibleStateSet 975
XAccessibleTable 971
XAccessibleText 969
XAccessibleValue 976
XAggregation 194, 201
XBookmarksSupplier 799, 806
XBoundComponent 896
XBreakIterator 385, 390
XBridgeFactory 78
XCalendar 384, 389
XCharacterClassification 384, 390
XChartData 710
XChartDataArray 710
XChild 903
XCollator 385, 391
XColumnsSupplier 806, 809, 833
XCommandPreparation 813, 847
XCommandProcessor 903
XCommandProcessor2 903
XCompletedConnection 798, 814
XComponent 100, 194, 200, 208
XComponentContext 33, 75, 84, 107, 121
XConnector 78
XContent 903
XContentCreator 903
XContentEnumerationAccess 85
XCustomPresentationSupplier 690
XDatabaseMetaData 848, 873
XDatabaseParameterBroadcaster 893
XDatabaseParameterListener 893
XDataDescriptorFactory 804
XDataSource 814
XDrawPageDuplicator 664
XDrawPages 664
XDrawPagesSupplier 706
XDriver 871
XDriverManager 815
XEnumerationAccess 95
XEventListener 97, 143, 167, 208
XExtendedCalendar 384
XExtendedIndexEntrySupplier 386
XFastPropertySet 91
XFlushable 799
XForm 890
XIndexAccess 95
XIndexContainer 95
XIndexEntrySupplier 386, 393
XInitialization 194, 200, 210
XInputSequenceChecker 386
XInterface 41, 89, 100, 194, 196, 203, 207, 220, 224
XLayerManager 685
XLoadListener 891
XLocaleData 384, 386
XMain 194, 201, 290
XML based filter 419
XML file format 419
XML filter adaptor 419
xml2cmp 178
XMultiComponentFactory 33, 75, 84, 107, 121
XMultiHierarchicalPropertySet 938
XMultiPropertySet 91
XMultiServiceFactory 84
XNameAccess 95
XNameContainer 95
XNamed 686
XNativeNumberSupplier 385, 392
XNumberFormatCode 385
XOutParameters 869
XPooledConnection 820
XPresentation 688f.
XPrintable 661
XPropertyContainer 903
XPropertySet 42, 91
XPropertySetInfo 91
XPropertyState 91, 888
XQueryDefinitionsSupplier 798, 800
XRefreshable 720
XRename 804
XResultSet 838
XResultSetAccess 828
XResultSetMetaData 845
XRow 836
XRowLocate 833
XRowSetApproveBroadcaster 826, 898
XRowSetApproveListener 826
XRowSetListener 826
XRowUpdate 840
XSelectionSupplier 702
XServiceInfo 194, 198f., 203, 221
XShapeCombiner 676

XShapeGrouper 676
XSQLQueryComposer 805
XSSOAcceptorContext 453
XSSOInitiatorContext 452
XSSOManager 452
XSSOManagerFactory 452
XStorable 658
XStyle 697
XTabControllerModel 890
XTableChartsSupplier 705, 709
XTablesSupplier 806, 808
XTransliteration 385, 392
XTypeProvider 194, 197, 203, 208, 220, 225
XUnoTunnel 194, 201
XUnoUrlResolver 33, 75, 107, 121
XUser 865
XViewDataSupplier 703
XWeak 100, 126, 194, 199, 203, 220
XWindowPeer 948
XYDiagram 708